View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Repositorio Universidad de Zaragoza

Universidad Zaragoza

Trabajo Fin de Grado

Desarrollo de concepto de un sistema de microfonía distribuido para sala de conferencias inalámbrico.

Autor

Iván Martín Escartín

Directores

José Ramón Beltrán Blázquez
Isidro Urriza Parroqué

Escuela de Ingeniería y Arquitectura
2014-2015

“Dedico este proyecto a todos los que
confiaron en mí, a mi familia, a mi chica,
a mis amigos y compañeros de la universidad,
y como no, a los profesores que me
apoyaron y ayudaron en todo momento,
haciendo que fuera posible terminar
esta etapa de mi vida”

TRABAJOS DE FIN DE GRADO / FIN DE MÁSTER

Escuela de Ingeniería y Arquitectura Universidad Zaragoza

DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD

(Este documento debe acompañar al Trabajo Fin de Grado (TFG)/Trabajo Fin de Máster (TFM) cuando sea depositado para su evaluación).

D./Dª. Iván Martín Escartín,

con nº de DNI 17760984 Q en aplicación de lo dispuesto en el art. 14 (Derechos de autor) del Acuerdo de 11 de septiembre de 2014, del Consejo de Gobierno, por el que se aprueba el Reglamento de los TFG y TFM de la Universidad de Zaragoza,

Declaro que el presente Trabajo de Fin de (Grado/Máster) Grado, (Título del Trabajo) Desarrollo de concepto de un sistema de microfonía distribuido para sala de conferencias inalámbrico,

es de mi autoría y es original, no habiéndose utilizado fuente sin ser citada debidamente.

Zaragoza, Junio 2015

Fdo: Iván Martín Escartín

Desarrollo de concepto de un sistema de microfonía distribuido para sala de conferencias inalámbrico.

RESUMEN

El proyecto aquí propuesto consiste en el desarrollo de un sistema de microfonía distribuido para salas de conferencia.

El sistema desarrollado se basa en programar y prototipar un dispositivo portátil el cual debe conectarse de manera inalámbrica, detectar automáticamente al hablante y gestionar el control de la instalación.

Una vez que el usuario del dispositivo habla por su micrófono el sistema reconoce automáticamente al hablante, manda inalámbricamente la señal y se reproduce en el resto de los dispositivos portátiles.

El sistema ha sido prototipado en el DSP TMX320C5535 de *Texas Instruments* utilizando el sistema de desarrollo C5535 eZDdsp y contiene los siguientes bloques y funciones:

Respecto a la parte hardware se pueden diferenciar cuatro partes: El códec de audio, el DSP C5535 de *Texas Instruments*, la UART y el módulo de comunicaciones inalámbrico.

En la parte software se encuentran las funciones programadas del DSP las cuales gestionan la configuración y el control de la instalación, además de las funciones del cálculo de la densidad espectral de potencia, el filtro para la banda telefónica y las funciones de compresión y descompresión de los datos.

ÍNDICE DE CONTENIDOS

Contenido

LISTADO DE FIGURAS

LISTADO DE TABLAS

LISTADO DE SÍMBOLOS Y ABREVIATURAS

DSP	*Digital Signal Processor*
DEP/PSD	Densidad espectral de potencia/Power Spectral Density
FFT	*Fast Fourier Transform*
UART	*Universal Asynchronous Receiver-Transmitter*
CCS	*Code Composer Studio*
ADC	*Analog Digital Converter*
DAC	*Digital Analog Converter*
FIFO	*First Input First Output*
IIR	*Infinite Impulse Response*
CR	*Carriage Return*
LF	*Line Feed*
SD	*Secure Digital*
PCB	*Printed Circuit Board*
UDP	*User Datagram Protocol*
SPI	*Serial Peripheral Interface*
PLL	*Phase-Locked Loop*
AT	*ATtention*
GPIO	*General Purpose Input Output*
COM	*COMmunications*
USB	*Universal Serial Bus*

MEMORIA

1

Introducción

En este capítulo se presenta la visión general del sistema que se ha desarrollado en este proyecto, la arquitectura general del mismo, los detalles generales de la implementación para finalizar con una visión general de la estructura de esta memoria.

1.1 Arquitectura del sistema

El trabajo desarrollado en este proyecto consiste en programar y prototipar un dispositivo de microfonía distribuido para salas de conferencia.

Las especificaciones del sistema que se ha desarrollado son: conectarse de manera inalámbrica con el resto de los sistemas portátiles, detectar automáticamente al hablante y gestionar el control de la instalación haciendo que no sea posible que más de un dispositivo esté transmitiendo a la vez.

La primera decisión que fue necesario tomar fue la plataforma en la que se iba a implementar este proyecto. El sistema propuesto es exigente desde el punto de vista del procesado de la señal de audio para la gestión de la instalación, por lo que se decidió realizar la programación sobre un DSP y no sobre otras plataformas como Arduino o Raspberry Pi.

De manera general el comportamiento del sistema de microfonía es el siguiente. Una vez que el sistema portátil detecta que existe un hablante, manda inalámbricamente la señal al resto de los dispositivos y la reproduce en ellos. Una vez que el hablante deja de hablar, el sistema debe ser capaz de detectarlo y volver al estado inicial. Se trata así de establecer una red de dispositivos sobre la sala de conferencia para que todos los participantes dispongan de un dispositivo portátil con micrófono y auriculares.

Para solventar el problema de la conectividad inalámbrica se ha decidido crear una red dedicada para los dispositivos gestionada con un router y que estos elementos transmitan en UDP y en *broadcast* a todos los elementos de la red. La elección de utilizar conectividad inalámbrica mediante WiFi radica en la amplia implantación de este tipo de redes de comunicaciones, lo que permitirá aumentar las características funcionales del sistema en un futuro.

El desarrollo del sistema ha sido pensado y diseñado en dos partes principales. La primera parte corresponde al desarrollo software que contiene todo lo necesario para la programación del DSP TMX320C5535 de *Texas Instruments* para realizar las funciones de procesado de señal y control de las comunicaciones. La segunda parte es el desarrollo del hardware necesario para que el sistema sea de tipo portátil y presente conectividad inalámbrica.

Los bloques presentes en el sistema son los siguientes:

Respecto a la parte hardware se pueden diferenciar cuatro partes: El códec de audio, el DSP C5535 de *Texas Instruments*, la UART y el módulo de comunicaciones inalámbrico.

En la parte software se encuentran las funciones programadas del DSP las cuales gestionan la configuración y el control de la instalación, además de las funciones del cálculo de la densidad espectral de potencia, el filtro para la banda telefónica y las funciones de compresión y descompresión de los datos.

La interacción entre los bloques del sistema se realiza de la siguiente manera. La señal captada por el micrófono es procesada por el filtro diseñado para la banda telefónica. Una vez obtenido el dato del micrófono se realiza un procesado de la señal para el cálculo de la DEP a

través de la FFT. Si la DEP es mayor que un cierto umbral el sistema reconoce que existe un hablante sobre el dispositivo, por lo que cambia al estado de envío en el cual los datos son transmitidos a través del módulo WiFi.

El resto de los dispositivos deben detectar que otro dispositivo les está mandando datos, y entrar en el modo de reproducción de los datos que les están llegando (el sistema da prioridad siempre a esta situación). Los datos recibidos también son procesados para que una vez que se detecte que ya no se está hablando el sistema vuelva al estado inicial.

El funcionamiento del sistema mediante un diagrama de flujo se puede observar en el apartado 4.1 *Diagrama de flujo del sistema*.

1.2 Implementación

La implementación del sistema ha sido desarrollada en el DSP de *Texas Instruments* TMX320C5535, programado en lenguaje C a través del programa CCS v5.3.0, utilizando la tarjeta de desarrollo C5535 eZdsp.

La implementación del sistema se ha llevado a cabo mediante una máquina de tres estados, a través de los cuales el sistema es capaz de gestionar completamente la instalación y cumpliendo la funcionalidad deseada. La máquina de estados se puede ver en la figura 1.

Los tres estados que se han implementado son los siguientes:

- Reposo: Estado inicial en el que el sistema da prioridad al dato que recibe a través de la UART (dato proveniente del módulo WiFi). En este estado el sistema comprueba si se están recibiendo datos provenientes de otros dispositivos. Si esto es así, el sistema pasa al estado *Reproducir*. Si no se reciben datos por la UART, se mira si se está hablando en el propio dispositivo procesando la señal para el cálculo de la DEP y comparándola con un cierto umbral. En el caso de que se esté hablando se vuelve a comprobar que no haya llegado en ese intervalo de tiempo ningún dato por UART, y si es así, se pasa al estado *Transmisión*. En caso de que no se reciban datos por la UART o no se esté hablando en el dispositivo, el sistema se queda en éste estado repitiendo las comprobaciones anteriores.
- Reproducir: Estado en el que se reproduce el dato que se está recibiendo por la UART. Además se hace un procesado del dato que se recibe para que, en el caso de estar recibiendo silencio (caso en el que dejan de hablar), se vuelve al estado *Reposo*.
- Transmisión: Estado en el que se está hablando por el dispositivo, por lo que el dato que se recibe del micrófono se manda por la UART para que llegue al módulo inalámbrico. Además se procesan los datos para que el sistema detecte cuando se deja de hablar, y volver al estado *Reposo*.

En el sistema prototipado se han incorporado indicadores luminosos para indicar en todo momento en qué estado se encuentra el sistema. La información que muestran los indicadores luminosos es la siguiente:

- Leds amarillo, azul, rojo y verde: Estado *Reposo*.
- Leds amarillo y azul: Estado *Transmisión*.
- Leds rojo y verde: Estado *Reproducir*.
- Led rojo: Realizando configuración del sistema.
- Leds apagados: Paso de un estado a otro.

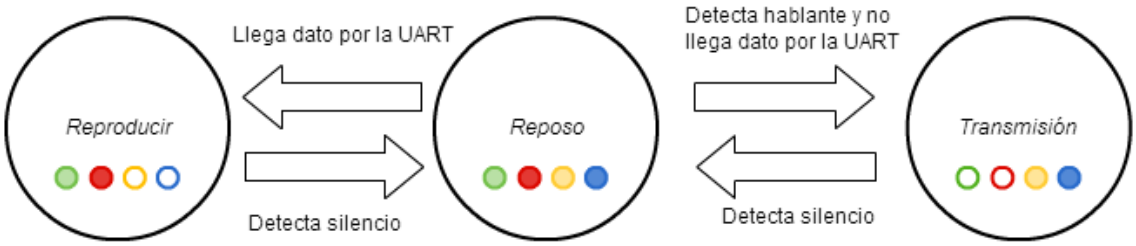

Figura 1: Máquina de estados

1.3 Estructura de la memoria

La memoria está estructurada de la siguiente manera:

- En el *capítulo 2. Configuración del sistema* se presentan las indicaciones necesarias para realizar la correcta configuración del DSP.

- En el *capítulo 3. Funciones de procesado de señal* se realiza una explicación de las funciones que se han tenido que programar como son: filtrado, cálculo de la densidad espectral de potencia, y compresión/descompresión.

- En el *capítulo 4. Diseño hardware y software* se explica detalladamente el funcionamiento del sistema que se ha desarrollado e implementado. Además se realiza el diseño de la placa de circuito impreso para hacer que el sistema sea de tipo portátil y presente conectividad inalámbrica. Por último, se aborda la comprobación del hardware y software mediante una revisión histórica de los pasos llevados a cabo hasta el desarrollo final.

- Por último en el *capítulo 5. Conclusiones* se hace una valoración de las características del sistema implementado, y se proponen una serie de mejoras que están abiertas como líneas de trabajo futuro.

2

Configuración del sistema

Hasta ahora se ha podido observar la arquitectura del sistema y el funcionamiento de éste. En este capítulo se aborda la configuración del sistema y de control tanto del DSP como de los periféricos necesarios para el arranque y el funcionamiento. Además también es necesaria la configuración de los módulos inalámbricos seleccionados.

La configuración de las diferentes secciones se ha realizado siguiendo las indicaciones del *TMS320C5535/34/33/32 Ultra-Low Power DSP Technical Reference Manual [1].*

Todo el sistema se ha desarrollado en la plataforma de desarrollo C5535 eZdsp la cual contiene además del DSP, una serie de periféricos y de interfaces de comunicación del DSP con el exterior. Como puede observarse en el Diagrama de bloques del 5535 eZdsp de la figura 2, es necesario configurar estos periféricos e interfaces de comunicación para el desarrollo de la aplicación, como son la UART, el bus I^2S el códec AIC3204 y el bus I^2C.

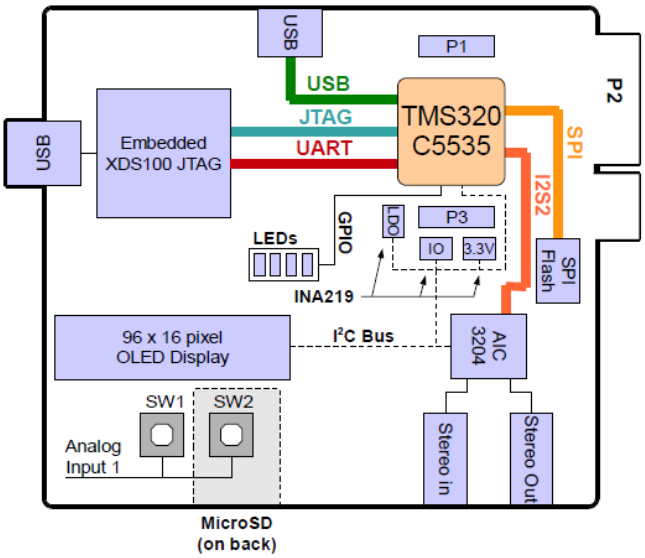

Figura 2: Diagrama de bloques del C5535 eZdsp

2.1 Configuración del sistema y control

Antes de crear el código de la aplicación lo primero que hay que hacer es realizar la configuración básica del sistema para que funcione correctamente. Ésta configuración contiene, por ejemplo, el reseteo de los periféricos, la habilitación del reloj sólo en los periféricos que se utilizan, la configuración del PLL y la selección del modo de funcionamiento del DSP.

Siguiendo las instrucciones del *Technical Reference Manual (Sección 1.7.5 Peripheral Reset) [1]* para resetear un grupo de periféricos se deben utilizar los registros *peripheral reset control register* (PRCR) y *peripheral software reset counter register* (PSRCR) y seguir los siguientes pasos:

- Poner como mínimo el valor 0x0008 en el registro PSRCR.
- Poner a 1 los periféricos que se quieren resetear en el registro PRCR.
- Esperar el número de ciclos necesario para que todos los periféricos se reseteen.

En algunos casos, como en el sistema desarrollado, un solo reset en el registro PRCR resetea varios periféricos a la vez. En el caso del sistema propuesto PG4_RST controla el reset del I2S2, I2S3, UART y del SPI.

Los registros comentados anteriormente pueden visualizarse en la imagen 3 y 4.

15							0
COUNT							
R/W-0							

LEGEND: R/W = Read/Write; *-n* = value after reset

Figura 3: Peripheral Software Reset Counter Register (PSRCR)

15	14	13	12	11	10	9	8
Reserved							
R-0							
7	6	5	4	3	2	1	0
PG4_RST	Reserved	PG3_RST	DMA_RST	USB_RST	SAR_RST	PG1_RST	I2C_RST
R/W-0	R-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; *-n* = value after reset

Figura 4: Peripheral Reset Control Register (PRCR)

Con estas instrucciones el código generado queda de la siguiente manera:

IO_PSRCR = 0x020; IO_PRCR = 0x0081; asm(" RPT #65535 "); asm(" NOP ");

Una vez reseteados los periféricos, y siguiendo el manual, se debe habilitar la señal de reloj para que llegue a los periféricos deseados. Esto se hace en los registros *Peripheral Clock Gating Configuration Registers* (PCGCR1/PCGCR2). Los cuales se muestran en las figuras 5 y 6.

15	14	13	12	11	10	9	8
SYSCLKDIS	I2S2CG	TMR2CG	TMR1CG	Reserved	TMR0CG	I2S1CG	I2S0CG
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
7	6	5	4	3	2	1	0
MMCSD1CG	I2CCG	Reserved	MMCSD0CG	DMA0CG	UARTCG	SPICG	I2S3CG
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; *-n* = value after reset

Figura 5: Peripheral Clock Gating Configuration Register (PCGCR1)

15							8
Reserved							
R-0							
7	6	5	4	3	2	1	0
Reserved	ANAREGCG	DMA3CG	DMA2CG	DMA1CG	USBCG	SARCG	LCDCG
R-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; *-n* = value after reset

Figura 6: Peripheral Clock Gating Configuration Register (PCGCR2)

El siguiente paso para la configuración es configurar el PLL, para ello se hace uso de los *Clock Generator Registers* (CGCR). Los cuales quedan explicados en *Technical Reference Manual (Sección 1.4.4) [1]* y pueden observarse en las figuras 7, 8, 9, 10.

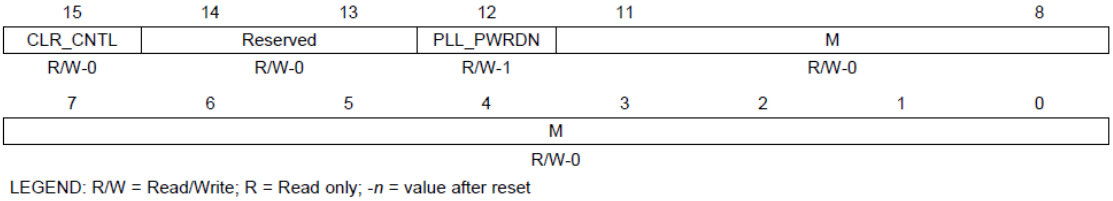

Figura 7: Clock Generator Control Register (CGCR1)

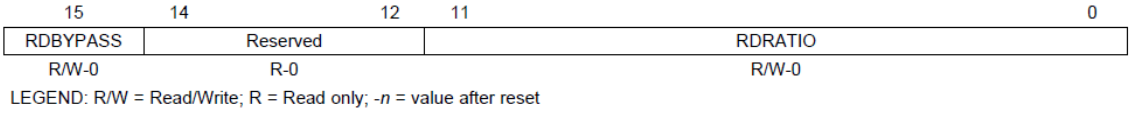

Figura 8: Clock Generator Control Register (CGCR2)

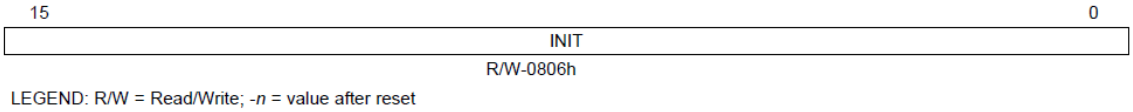

Figura 9: Clock Generator Control Register (CGCR3)

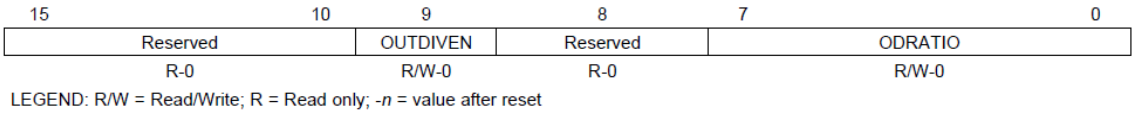

Figura 10 Clock Generator Control Register (CGCR4)

Quedando la configuración del sistema de la siguiente manera:

PLL_CNTL1 = 0x8BE8; PLL_CNTL2 = 0x8000; PLL_CNTL3 = 0x0806; PLL_CNTL4 = 0x0000;

Por ultimo para terminar la configuración se debe de configurar el mapeo de las señales I2S2, I2S3, UART, SPI y GPIO a los pines a través del registro *External Bus Selection Register* (EBSR). El registro contiene los campos que se muestran en la figura 11 y quedan configurados de la siguiente manera:

IO_EBSR = 0x1A00;

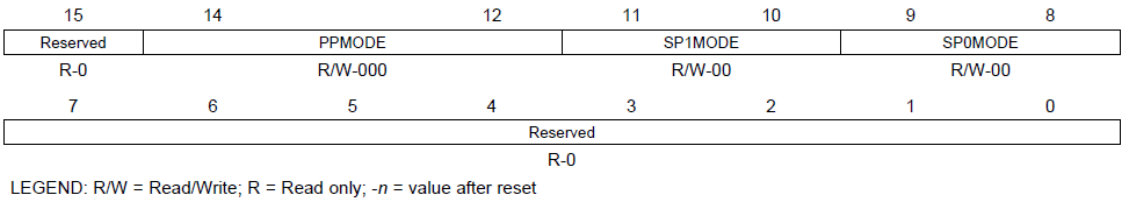

Figura 11: External Bus Selection Register (EBSR)

Una vez configurado el funcionamiento deseado del DSP ya se puede pasar a la configuración de los periféricos, la cual se describe en los apartados siguientes.

2.2 UART

La UART proporciona una conversión serie-paralelo de los datos que se reciben y una conversión paralelo-serie para los datos transmitidos. Además permite entre otras cosas configurar la tasa de bits (*baud rate)* deseada para establecer diferentes velocidades de comunicación, cambiar el tamaño de los bits de datos, decidir si se quiere paridad o no, cuantos bits de stop se desean, etc.

Tal y como indica *Technical Reference Manual (Capítulo 7) [1]* para configurar la UART se deben de configurar sucesivamente una serie de registros, los cuales quedan reflejados en la tabla 1.

RBR	*Receiver Buffer Register*
THR	*Transmitter Holding Register*
IER	*Interrupt Enable Register*
IIR	*Interrupt Identification Register*
FCR	*FIFO Control Register*
LCR	*Line Control Register*
MCR	*Modem Control Register*
LSR	*Line Status Register*
SCR	*Scratch Register*
DLL	*Divisor LSB Latch Register*
DLH	*Divisor MSB Latch Register*
PWREMU_MGMT	*Power and Emulation Management Register*

Tabla 1: Registros de la configuración UART

Para que el sistema de transmisión funcione correctamente la UART debe adaptarse a los módulos inalámbricos. Estos módulos suelen presentar un bit de inicio (*start)*, 8 bits de datos y un bit de parada (*stop)*, lo que implica que un símbolo se transmite con 10 bits.

Como el sistema trabaja con datos de 16 bits se ha implementado la compresión a 8 bits de los datos capturados por el micrófono para enviarlos a través de la UART.

La tasa de bits (*baud rate*) seleccionada es de 115200 baudios debido a que el mínimo para poder trabajar con audio en tiempo real son 8kHz que se corresponden con 80000 baudios. Para configurar la UART del DSP a esta velocidad se deben realizar los siguientes pasos:

- Los registros DLL y DLH contienen el *baud rate* que se desea establecer en la comunicación. Siguiendo la fórmula que establece *Technical Reference Manual (Sección 7.2.1) [1]*, para obtener un *baud rate* de 115200 baudios se debe de poner un valor entero de 54.
- El registro FCR contiene el modo FIFO para los datos que se transmiten o se reciben.

- El registro LCR contiene la configuración de la trama de datos, la cual debe ser la misma que la que acepta el módulo WiFi, por lo que debe de ser configurada con 8 bits de datos, un bit de *start*, un bit de stop, sin paridad.
- El registro MCR habilita que las señales internas de la UART estén disponibles en los pines exteriores de la placa de desarrollo del DSP.
- El registro PWREMU_MGMT contiene la habilitación de la transmisión/recepción así como del estado de reset.

Para la configuración del UART como indica *eZdsp™ Technical Reference [2]* se debe configurar el switch3 presente en la placa (ver figura 12) tal y como indica la figura 13, ya que esto deshabilita el UART por el conector J2.

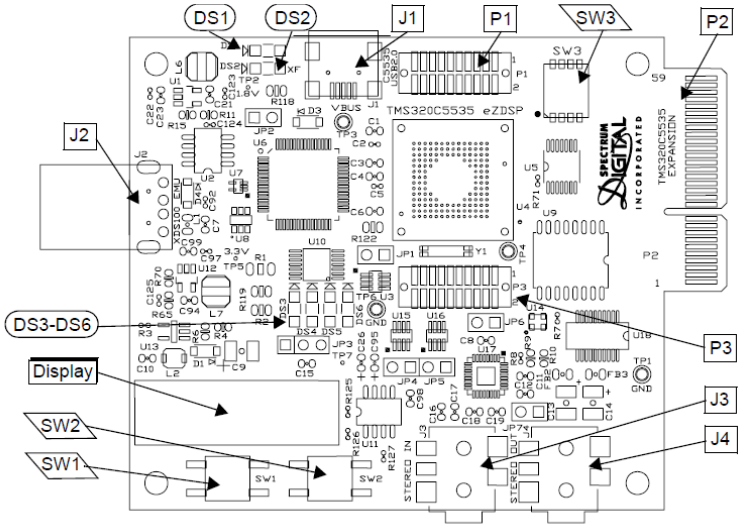

Figura 12: C5535 eZdsp (top)

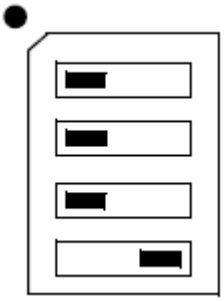

Figura 13: Switch3

La configuración de la UART en el código de la aplicación ha sido integrada en una función llamada *UART_init()* (ver Anexo I.6 *Uart.c*) la cual asigna los valores adecuados a los registros comentados anteriormente.

2.3 Bus de control I2C

Como indica *Technical Reference Manual (Capítulo 9) [1]* se trata de un bus creado para conectar periféricos a un microcontrolador a baja velocidad (<100kb/s) que presenta una arquitectura de dos cables, uno de datos y otro de reloj. Los dispositivos conectados al bus son identificados con una dirección única y pueden actuar como maestros o como esclavos independientemente de que operen como transmisores o como receptores.

Los registros que se pueden modificar son los mostrados en la tabla 2:

ICOAR	*I2C Own Address Register*
ICIMR	*I2C Interrupt Mask Register*
ICSTR	*I2C Interrupt Status Register*
ICCLKL	*Clock Low-time*
ICCLKH	*Clock High-time*
ICCNT	*I2C Data Count Register*
ICDRR	*I2C Data Receive Register*
ICSAR	*I2C Slave Address Register*
ICDXR	*I2C Data Transmit Register*
ICMDR	*I2C Mode Register*
ICIVR	*I2C Interrupt Vector Register*
ICEMDR	*I2C Extended Mode Register*
ICPSC	*I2C Prescaler Register*
ICPID1	*I2C Peripheral Identification Register*
ICPID2	*I2C Peripheral Identification Register*

Tabla 2: Registros de la configuración del bus I2C

La inicialización del bus se realiza mediante la función *USBSTK5515_I2C_init( )* (ver Anexo I.7 *Usbstk5515_i2c.c*) la cual configura adecuadamente los registros nombrados en la tabla 2 para el funcionamiento del bus. Debido a que la configuración del bus I^2C es siempre la misma se ha utilizado la función que proporciona *Texas Instruments* para la configuración del bus con los parámetros adecuados.

2.4 Códec TLV320AIC3204

Para procesar la señal de audio proveniente de un sistema de captación analógico como es el micrófono es necesario utilizar un conversor analógico-digital. Así mismo para la reproducción del audio del dato que ha sido procesado es necesario un conversor digital-analógico. Hay que destacar que las conversiones analógico-digital y digital-analógico son operaciones inversas si se cumple el teorema del muestreo de Nyquist-Shannon salvo por el error de cuantificación.

El teorema de Nyquist-Shannon indica que la frecuencia de muestreo debe ser mayor o igual a dos veces la frecuencia máxima de la señal (normalmente el ancho de banda). Como la señal de voz suele presentar la información relevante hasta los primeros 4 kHz, la frecuencia de muestreo mínima debe de ser al menos 8kHz. No obstante en aplicaciones de audio profesional se suele utilizar una frecuencia de muestreo de 48kHz ya que está por encima del doble de la frecuencia máxima audible por el oído humano que es de 20kHz.

Los conversores AD y DA están presentes en el códec TLV320AIC3204, y deben ser configurados correctamente. En este caso se ha configurado una frecuencia de muestreo de 48kHz, datos de 16 bits, dos canales en configuración *mono*, y se ha modificado la ganancia de

los conversores con el fin de ajustarse al micrófono y a los auriculares para intentar obtener la mayor amplitud de la señal de salida sin distorsión.

La configuración del códec se realiza a través del bus serie I^2C que conecta el DSP y el códec, mientras que los datos de audio se intercambian a través del bus serie I^2S2 tal y como indica la figura 3 Diagrama de bloques del C5535 eZdsp.

Los registros del códec han sido configurados siguiendo las instrucciones de *TLV320AIC3204 Ultra Low Power Stereo Audio Codec [3]* y *TLV320AIC3204 Application Reference Guide [4].*

En el sistema desarrollado se ha creado una función llamada *AIC3204_1_init()* (ver Anexo I.2 *Aic3204_1.c*) la cual realiza la configuración del códec, la configuración de los relojes, la configuración del DAC, la configuración del ADC y la configuración del I2S2.

2.5 Módulos inalámbricos

Para solventar el problema de la conectividad inalámbrica, se ha optado por la elección de un módulo WiFi conectado a un DSP a través de UART. Estos módulos se suelen programar mediante comandos AT, y se pueden conectar mediante UART a otro dispositivo.

Para las pruebas se ha utilizado un módulo como es el ESP8266. La información relativa a la programación del módulo se encuentra disponible en el sitio web https://nurdspace.nl/ESP8266. Para su adecuada puesta en marcha se ha actualizado la versión del firmware a la versión v0.9.2.2. Una vez actualizado se ha configurado mediante Arduino a velocidad de 115200 baudios, ya que al actualizar la versión del software la velocidad por defecto es de 9600 baudios, y hay que mandarle a esta velocidad el comando AT+CIOBAUD=115200 para que se cambie la velocidad. Esto es debido a que la UART del DSP está programada a esta velocidad de 115200 y no es posible mandar un comando a otra velocidad menor.

Para la familiarización con dichos módulos se ha utilizado la plataforma Arduino, ya que presenta una interfaz más agradable y usable que la programación directa en lenguaje C. Una vez conocido su funcionamiento se programaron en lenguaje C las funciones que establecían las configuraciones tanto de modo servidor (modo por defecto en el que los módulos WiFi escuchan si alguien les está mandando algo), modo cliente (modo al que pasa cuando quieren transmitir al resto de los dispositivos), y conexión al punto de acceso (conexión a la red creada exclusivamente para éstos dispositivos).

Las diferentes funciones que se implementaron fueron: *conectar(), configurar_cliente()* y *configurar_servidor()* (ver Anexo I.1 *Main.c*).

En el sistema final no ha sido posible implementar estos módulos debido a que en la recepción de los datos introducen unas cabeceras, las cuales deben ser eliminadas antes de reproducir el dato, por lo que se debería incrementar la velocidad para poder procesar los datos correspondientes a 8kHz. Una prueba que se ha realizado sin éxito ha sido aumentar la velocidad de los módulos a 921600 baudios (velocidad máxima que admite el módulo

ESP8266) ya que la cabecera son 9 símbolos para intentar procesarla y reproducir el dato de audio a 8kHz. Además otro problema que se ha encontrado es que el DSP en la recepción de la UART tiene una memoria FIFO la cual tiene un tamaño máximo de 16 palabras, por lo que el empaquetado de los datos queda bastante limitado, ya que no se pueden almacenar más de 16 símbolos.

Actualmente no existen en el mercado módulos WiFi que no incorporen cabeceras en los datos recibidos. Si hubiera módulos que transmitieran y recibieran datos en bruto con velocidad de 115200 baudios sería suficiente para poder implementar este sistema.

Otra alternativa a los módulos WiFi sería la utilización de cualquier módulo de comunicaciones inalámbricas el cual permitiera el envío y recepción de datos en tiempo real a la velocidad adecuada, como podría ser un *transceiver* de RF. Estos módulos inalámbricos de RF permitirían el envío y recepción de los datos en bruto pero no ofrecerían las ventajas que incorpora el utilizar un protocolo tan extendido a día de hoy como es el protocolo TCP/IP.

2.6 Carga del programa mediante tarjeta SD

Una desventaja de utilizar el DSP C5535 es que el código de la aplicación se pierde cada vez que se desconecta de la alimentación. El sistema final ha sido desarrollado con el fin de cargar en una tarjeta SD el programa generado y que mediante un lector de tarjetas SD el DSP sea capaz de ejecutar el código cada vez que se conecta la alimentación, lo que hace posible el prototipado portátil del sistema.

Una vez que mediante CCS v5 se ha generado el fichero de compilación .out, se debe ejecutar en consola el fichero hex55.exe con el siguiente comando:

```
>> hex55 -i Aplicacion.out -o bootimg.bin -boot -v5505  -b -serial8
```

Este comando genera un fichero llamado bootimg.bin el cual contiene en binario el código de la aplicación. El fichero bootimg.bin debe de ser copiado en la tarjeta SD para que el DSP lo ejecute cuando se conecte la alimentación.

En este capítulo se ha realizado una visión de la configuración del sistema observando en los diferentes apartados la configuración de cada una de las partes del sistema.

Durante los capítulos posteriores se llevará a cabo una visión de las funciones implementadas, el diseño software del sistema global y el diseño hardware necesario para que el sistema sea de tipo portátil y presente conectividad inalámbrica.

3

Funciones de procesado de señal

Se han desarrollado tres funciones de procesado de señal necesarias para el control del hablante, la limitación en banda de la señal de audio y la compresión/descompresión necesaria para la transmisión de los datos a través de la UART.

Las funciones son: Filtro para la banda telefónica, Cálculo de la densidad espectral de potencia, compresión/descompresión y se detallan a continuación.

3.1 Filtro para la banda telefónica

La señal de audio proveniente del micrófono no sólo contiene la información deseada que es la señal de voz, además puede contener ruido e interferencias provenientes de otras fuentes sonoras las cuales intervienen negativamente en nuestro sistema.

Para eliminar en mayor medida la contribución de estas partes negativas a la señal de audio, se debe realizar un filtrado para la señal de voz en la banda frecuencial en la que ésta se encuentra.

Un filtro es un componente activo o pasivo el cual, mediante procesamiento matemático de la señal de entrada, permite obtener a su salida el resultado de dicha operación con el objetivo de resaltar o atenuar ciertas características de la señal. En otras palabras, es un sistema que modifica el contenido frecuencial de la señal de entrada de forma preestablecida.

La señal de voz presenta la información relevante aproximadamente desde los 50Hz hasta los 4kHz. Por ello mismo, se ha diseñado un filtro el cual permite quedarse con la señal deseada, sin apreciar apenas distorsión de la señal y permitiendo la identificación posterior del hablante.

El filtro que se ha implementado es un filtro IIR (la salida actual depende, además de las muestras pasadas y actuales de entrada, de las muestras de salida anteriores) en secciones de orden 2 colocadas en cascada con el fin de conseguir que sea estable. Cada sección de orden 2 presenta el esquema que se indica en la figura 14.

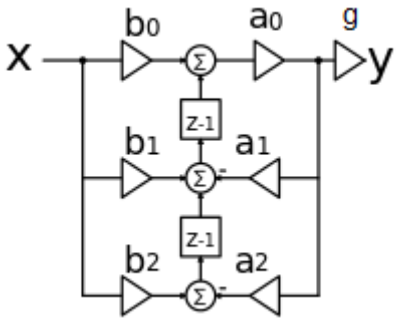

Figura 14: Sección de orden 2 del filtro

El filtro ha sido implementado en digital, ya que el procesamiento de la señal digital es mucho más sencillo y mucho más fácil de llevar a la práctica que uno en el dominio analógico. Esto se debe a que la implementación de sistemas digitales cubre un amplio margen de posibilidades. No obstante debe existir un filtro analógico *antialiasing* a la entrada del conversor analógico digital presente en forma de condensador para cumplir el teorema de Nyquist en todo momento.

En el caso de este trabajo se requiere que el filtrado sea en tiempo real por lo que el sistema debe de ser causal y estable.

Antes de crear en lenguaje C el filtro, se ha simulado en Matlab mediante la herramienta de simulación *fdatool* y se han comparado los resultados obtenidos en ambos casos.

Las especificaciones del filtro han sido las siguientes:

- Respuesta tipo paso banda.
- Tipo IIR
- Frecuencia de muestreo de 48kHz.
- Frecuencias de corte entre 300-500Hz y 3-4kHz.
- Atenuación con respecto a la banda de paso de 50dB
- Rizado de 1 dB en la banda de paso.

Con estas especificaciones, el filtro obtenido en Matlab se puede observar en la figura 15.

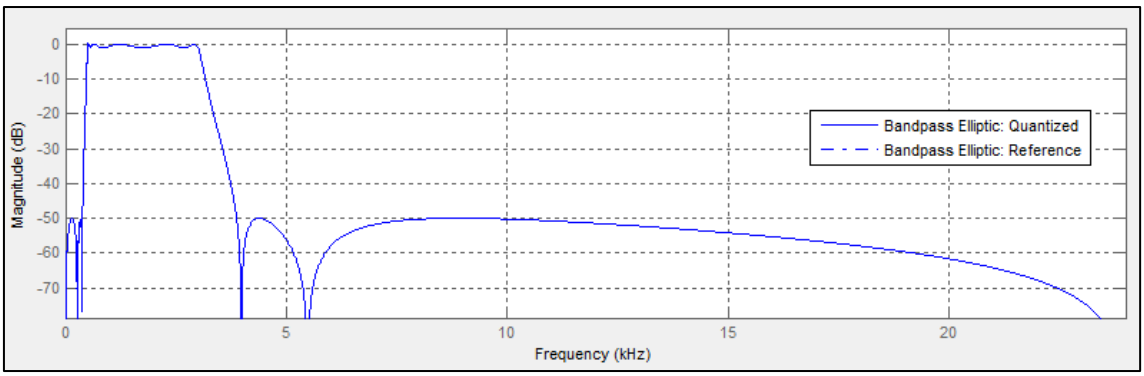

Figura 15: Respuesta en frecuencia del filtro paso banda.

Además del cálculo de los coeficientes, la herramienta *fdatool* permite la visualización de la respuesta al impulso, así como el diagrama de polos y ceros del filtro los cuales pueden ser observados en la figura 16 Respuesta al impulso y 17 Diagrama de polos y ceros del filtro.

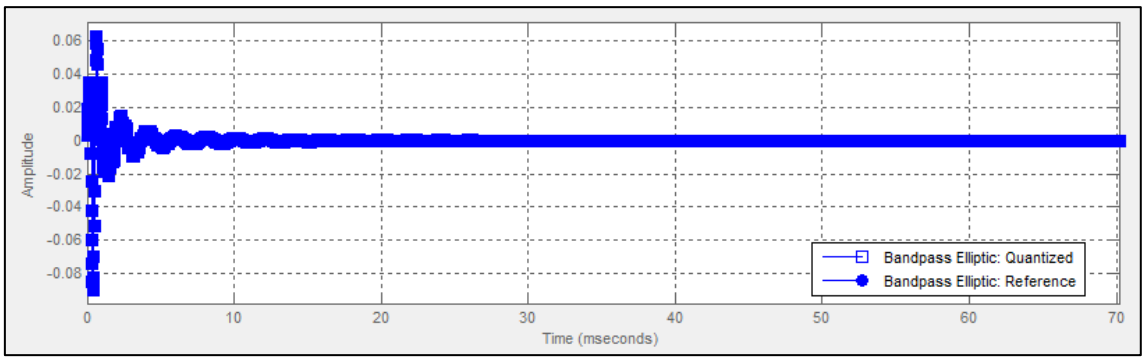

Figura 16: Respuesta al Impulso

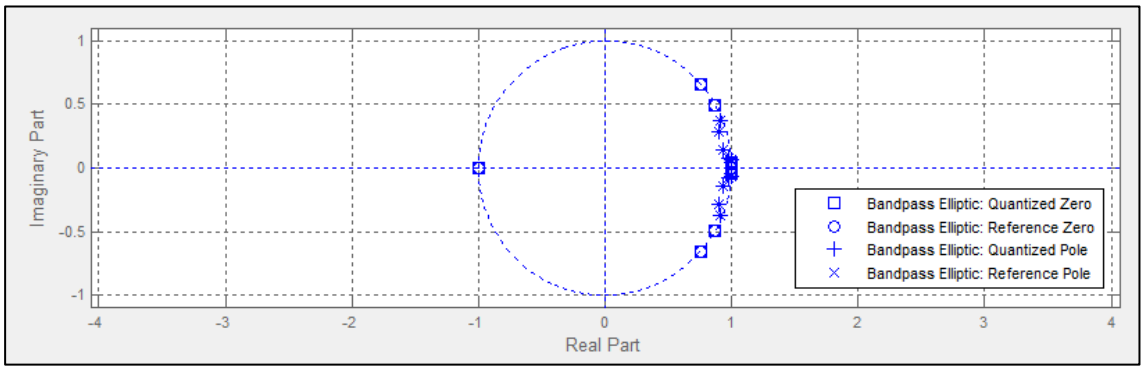

Figura 17: Diagrama de polos y ceros del filtro

Una vez obtenido el filtro se ha pasado a la cuantización de los coeficientes del filtro para observar el comportamiento una vez codificados en coma fija. Al simular éste comportamiento se ha observado que el desplazamiento que se realizaría en lenguaje C al ajustar las escalas introduce una señal continua al desplazar hacia arriba la cuantización, efecto que se puede observar en la figura 18. Para solventar éste problema, en vez de realizar un truncado directamente, se ha implementado el redondeo y posteriormente el truncado del dato.

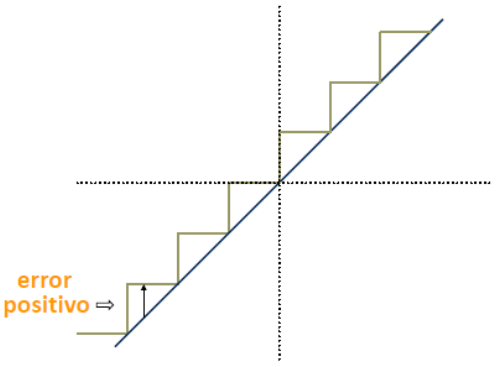

Figura 18: Efecto del truncado en la cuantización

Una vez simulado el comportamiento del filtro en Matlab de las diferentes secciones y del comportamiento global del sistema se pueden visualizar los resultados en las figuras 19 y 20.

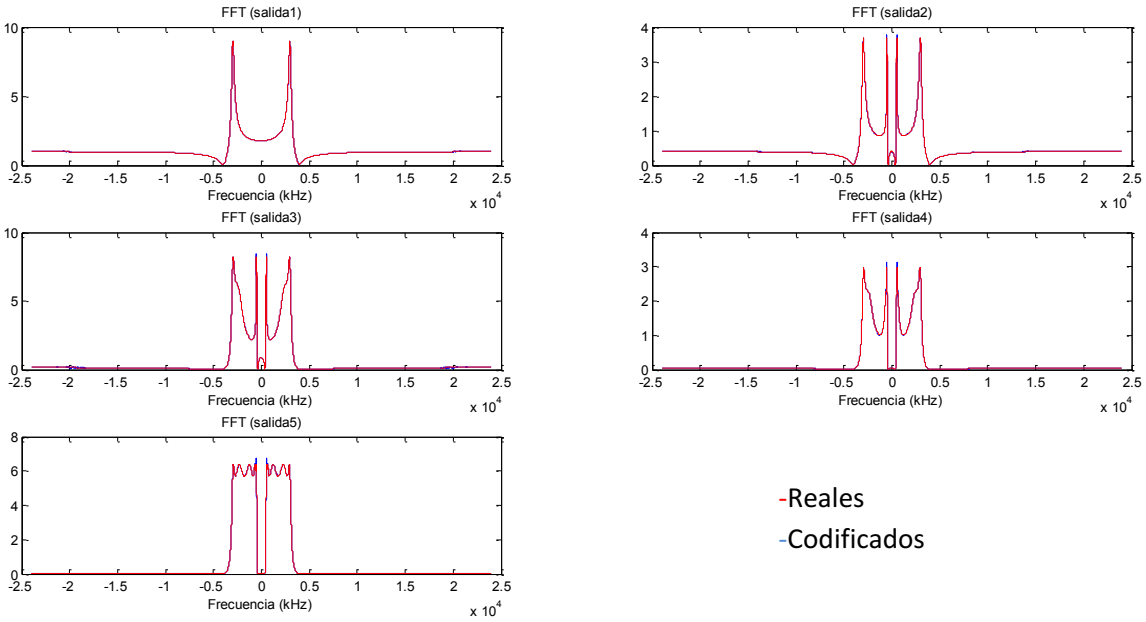

Figura 19: Aspecto en el dominio frecuencial de la respuesta impulsional teórica de las secciones de orden 2

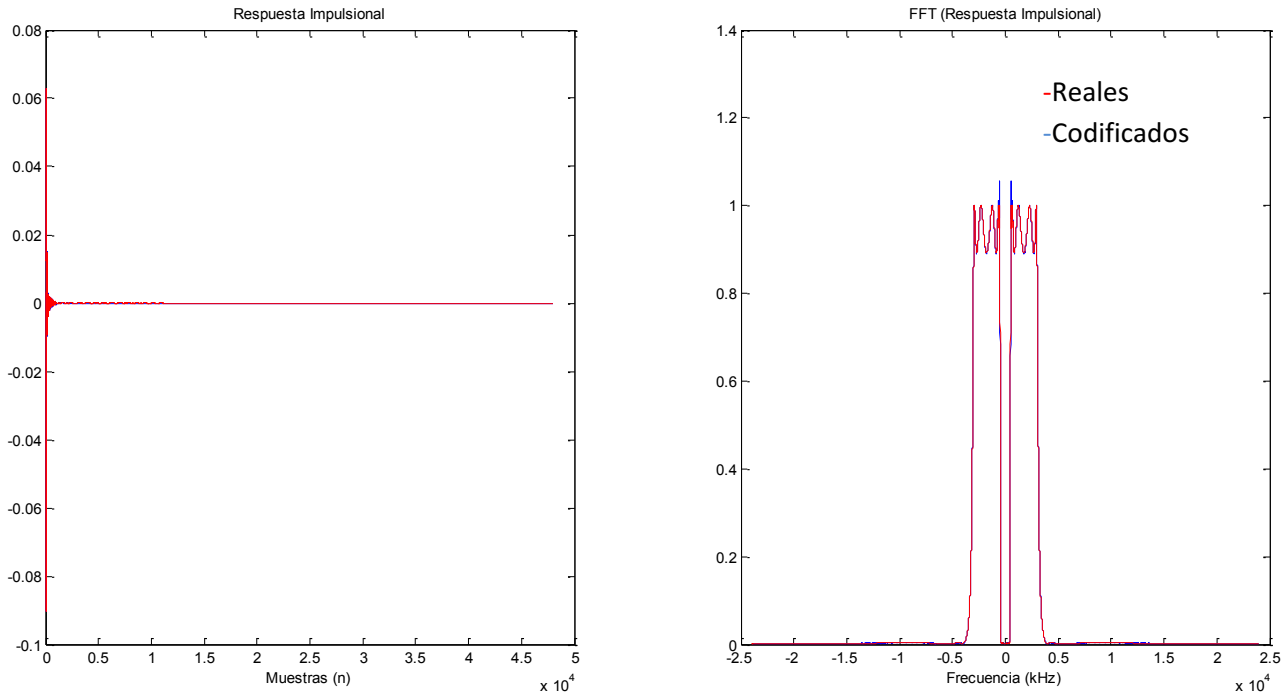

Figura 20: Respuesta impulsional y Respuesta frecuencial del filtro en conjunto

Una vez comprobado el correcto funcionamiento del sistema en Matlab, se ha procedido a la implementación en lenguaje C del sistema, con los coeficientes del filtro observados en la tabla 3. Posteriormente se ha calculado la respuesta impulsional del sistema al pasar una delta por el filtro en C y los resultados se han trasladado a Matlab para poder visualizarlos, obteniendo los resultados observados en las figuras 21 y 22. Observando que se cumplen los requisitos de las especificaciones del sistema.

	B0	B1	B2	A0	A1	A2	Ganancia
Etapa1	32767	-28416	32767	32767	-29772	31662	26586
Etapa2	32767	-32729	32767	32767	-32601	32576	26586
Etapa3	32767	-24687	32767	32767	-29437	29146	22634
Etapa4	32767	-32748	32767	32767	-32160	31768	22634
Etapa5	32767	0	-32768	32767	-30587	29208	20464

Escala
<16.14>
<16.15>
<16.16>
<16.17>

Tabla 3: Coeficientes codificados del filtro

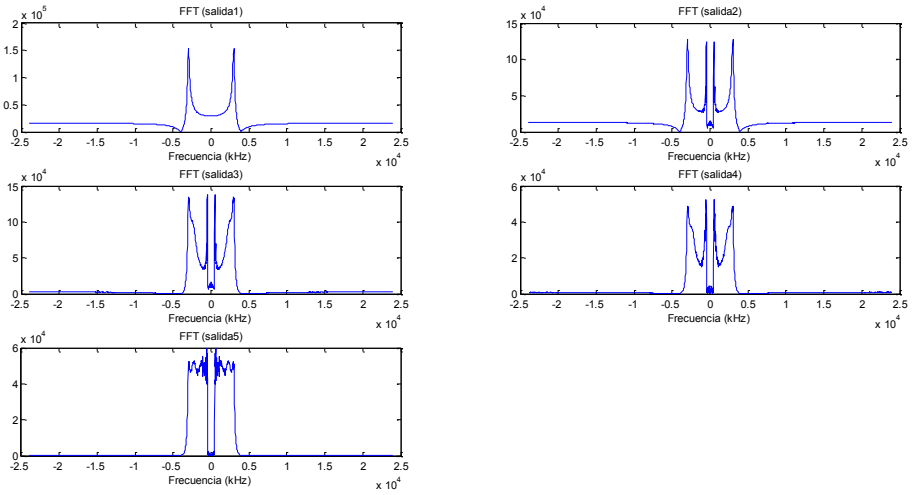

Figura 21: Aspecto en el dominio frecuencial de la respuesta impulsional real de las secciones de orden 2 en lenguaje C

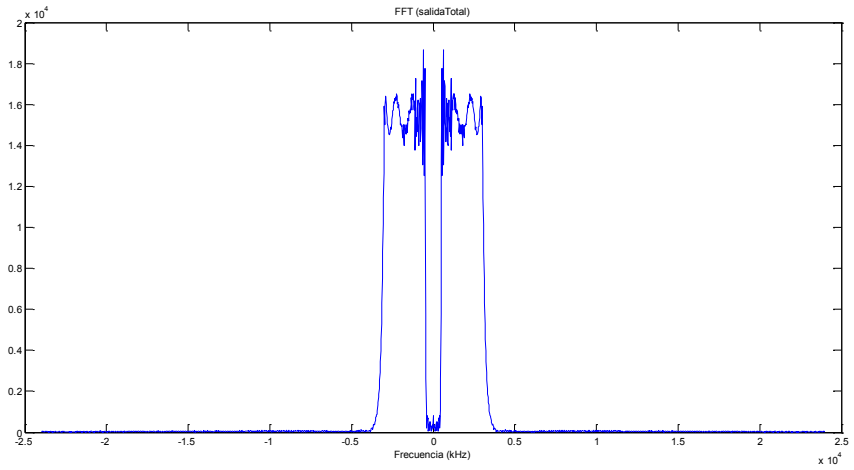

Figura 22: Respuesta frecuencial real del filtro en conjunto en lenguaje C

Por último se ha medido el tiempo en ciclos de reloj que le cuesta ejecutar la tarea del filtrado en el proyecto que es de 221 ciclos/muestra.

3.2 Calculo de la Densidad Espectral de Potencia

La densidad espectral de potencia indica cómo está distribuida la potencia de la señal en el espectro frecuencial. Se ha optado utilizar este procesado tras pasar el dato recibido por el filtro de la banda telefónica con lo que se obtiene únicamente la señal que se encuentra dentro del rango frecuencial de la voz, con el fin de conocer cuándo existe una persona hablando o cuándo no.

Si el cálculo de la densidad espectral de potencia es menor que un cierto umbral, el sistema detectará que no existe ningún hablante mientras que si se supera un cierto umbral establecido con anterioridad el sistema detectará que existe un hablante y pasará a la transmisión de los datos.

El cálculo de la densidad espectral se puede calcular como el cuadrado del módulo de la Transformada de Fourier de la señal tal y como indica la ecuación:

$$DEP = \int_{-\infty}^{\infty} lim_{T\to\infty} \left(\frac{|X(f)|^2}{T}\right) df$$

Para el cálculo de la FFT se debe acudir a *Technical Reference Manual (Capítulo 2) [1]* y seguir las instrucciones que este indica con el fin de realizar correctamente el procesado de la señal.

No obstante se ha consultado la *Silicon Revision 2.2 [5]* ya que existía un error en la versión *Technical Reference Manual (Capítulo 2) [1].*

Existen dos maneras diferentes para realizar el cálculo de la FFT, la primera es que el TMS320C5535 contiene un acelerador por hardware que acelera el cálculo de la FFT, y la segunda es el cálculo por software.

Durante las pruebas realizadas en el diseño se utilizó tanto el acelerador hardware como el cálculo por software. No obstante, el acelerador por hardware no ha sido posible utilizarlo en el sistema final ya que no existe suficiente documentación de los registros que habilitan dicho acelerador al hacer un programa que arranque desde cero, por lo que finalmente se ha utilizado el cálculo mediante software.

Para que el sistema realice la FFT se debe incluir en la ruta de directorios (*path*) de búsqueda del proyecto la carpeta que contiene la librería del cálculo de la FFT. Además tal y como indica *Technical Reference Manual (Capítulo 2) [1]* se deben alinear ciertos registros implicados en el cálculo de la FFT mediante el comando *#pragma* para poder utilizar correctamente las funciones *cfft()* y *cbrev()*.

El resultado de la FFT se almacena en un registro de la longitud indicada al número de puntos con el que se ha realizado la FFT de 32 bits cada uno. En el que los 16 primeros bits representan la parte real y los 16 bits siguientes representan la parte imaginaria.

Para el cálculo de la densidad espectral se debe separar la parte real y la imaginaria y posteriormente realizar el cálculo del módulo al cuadrado.

La función que realiza el cálculo de la densidad espectral en el código desarrollado es: *calculo_PSD()*(ver Anexo I.1 *Main.c*).

Una vez acabada la función se ha calculado cuanto tiempo en ciclos de reloj tarda en realizarse el cálculo de la DEP obteniendo un total de unos 160000 ciclos de reloj.

3.3 Compresión y descompresión

Debido a que el dato obtenido del micrófono se codifica en una variable de tipo entero de 16 bits y los módulos inalámbricos utilizan datos de 8 bits, se ha incorporado el método de compresión-expansión (*companding)* con el fin de obtener el dato original transmitiendo solamente una vez cada dato.

La ley que se ha seguido es la que se utiliza en Europa, la ley-A. Los motivos por lo que se ha decantado por este tipo de algoritmo es que se utiliza principalmente para la voz humana ya que el funcionamiento explota las características de ésta.

Éste método expande las amplitudes de la señal de audio pequeñas mientras que las de amplitudes más elevadas se comprimen ya que tienen menos posibilidad de aparición. Es decir, se aplica una cuantificación no uniforme (logarítmica) a la señal original, en la cual se obtienen pasos de cuantificación pequeños para los valores pequeños de amplitud y pasos grandes para los valores grandes.

Dicho algoritmo es un mecanismo de compresión con pérdidas ya que no se recupera exactamente la señal original.

Siguiendo *TMS320C6000 µ-Law and A-Law Companding [6]* en la práctica la cuantificación logarítmica se ha llevado a cabo aproximando la función logarítmica por segmentos de rectas, los cuales se aproximan fielmente a la logarítmica pero simplifican la cuantización como se puede observar en la figura 23.

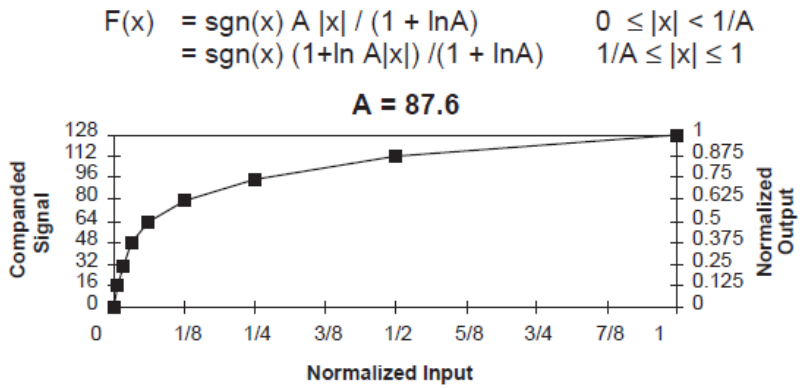

Figura 23: Aproximación lineal a la ecuación de compresión logarítmica Ley-A

La ley-A utiliza palabras de entrada de 13 bits, en los que el bit más significativo es el bit de signo. Posteriormente se codifican el resto de los bits siguiendo la tabla de compresión de la figura 24.

Input Values													Compressed Code Word						
													Chord			Step			
Bit:	11	10	9	8	7	6	5	4	3	2	1	0	Bit:	6	5	4	3	2	1	0
	0	0	0	0	0	0	0	a	b	c	d	x		0	0	0	a	b	c	d
	0	0	0	0	0	0	1	a	b	c	d	x		0	0	1	a	b	c	d
	0	0	0	0	0	1	a	b	c	d	x	x		0	1	0	a	b	c	d
	0	0	0	0	1	a	b	c	d	x	x	x		0	1	1	a	b	c	d
	0	0	0	1	a	b	c	d	x	x	x	x		1	0	0	a	b	c	d
	0	0	1	a	b	c	d	x	x	x	x	x		1	0	1	a	b	c	d
	0	1	a	b	c	d	x	x	x	x	x	x		1	1	0	a	b	c	d
	1	a	b	c	d	x	x	x	x	x	x	x		1	1	1	a	b	c	d

Figura 24: Tabla de compresión de la Ley-A

El dato original de la señal de audio está representado con 16 bits por lo que se escogen los 13 bits más significativos como palabra de entrada a la tabla de la figura anterior.

En el proceso de recepción se debe realizar el procesado inverso del dato con el fin de obtener el original salvo por las pérdidas de cuantificación. La tabla que representa éste proceso queda reflejada en la figura 25.

$$F^{-1}(y) = \operatorname{sgn}(y)\,|y|\,[1+\ln(A)]/A, \qquad 0 \le |y| \le 1/(1+\ln(A))$$
$$= \operatorname{sgn}(y)\,e^{(|y|[1+\ln(A)]-1)}/[A+A\ln(A)], \qquad 1/(1+\ln(A)) \le |y| \le 1$$

Compressed Code Word								Biased Output Values											
Chord				Step															
Bit:	6	5	4	3	2	1	0	Bit:	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	0	a	b	c	d		0	0	0	0	0	0	0	a	b	c	d	1
	0	0	1	a	b	c	d		0	0	0	0	0	0	1	a	b	c	d	1
	0	1	0	a	b	c	d		0	0	0	0	0	1	a	b	c	d	1	0
	0	1	1	a	b	c	d		0	0	0	0	1	a	b	c	d	1	0	0
	1	0	0	a	b	c	d		0	0	0	1	a	b	c	d	1	0	0	0
	1	0	1	a	b	c	d		0	0	1	a	b	c	d	1	0	0	0	0
	1	1	0	a	b	c	d		0	1	a	b	c	d	1	0	0	0	0	0
	1	1	1	a	b	c	d		1	a	b	c	d	1	0	0	0	0	0	0

Figura 25: Tabla de descompresión de la Ley-A

En el código del proyecto existen dos funciones las cuales realizan el proceso de *compandig*. Éstas funciones son *compresión()* y *descompresión()* (ver Anexo I.3 *Compresion.c* y I.4 *Descompresion.c*).

3.4 Envío por la UART

La configuración de los módulos inalámbricos suele ser a través de comandos AT, por lo que se han implementado funciones que simplifican el envío de éstos comandos.

Se ha creado en lenguaje C una función a la que le se le pasa como parámetro un *string* de caracteres con el comando a enviar terminado en \015\012 los cuales representan a los comandos CR y LF en octal.

Esta función envía por la UART a la velocidad configurada los caracteres del *string* con el fin de simplificar la configuración de los módulos inalámbricos. Además entre dos comando sucesivos de configuración de los módulos es necesario esperar un cierto tiempo para que el módulo se configure correctamente y responda, por lo que también se ha implementado una función que espera un cierto tiempo de ciclos de reloj.

En el código las funciones que realizan estas operaciones son las funciones: *enviar()* y *esperar()* (ver Anexo I.1 *Main.c*).

4

Diseño hardware y software

Hasta este capítulo se ha realizado la configuración del DSP y los periféricos, así como de las funciones necesarias para la programación del código de la aplicación.

En este capítulo se presentan los detalles de implementación del sistema. Para la depuración del código en lenguaje C se ha utilizado el entorno de programación CCS.

En el capítulo se explica mediante un diagrama de flujo el funcionamiento del sistema. El código generado en éste capítulo se puede ver en el Anexo I: Código lenguaje C.

Una vez presentada la parte del desarrollo de software, se presenta el diseño hardware que ha sido necesario realizar en este trabajo.

Una de las características que debe tener el sistema es que sea un dispositivo portátil, además de presentar conectividad inalámbrica. Se ha optado por el diseño de una placa de circuito impreso que contenga por un lado, el diseño de la alimentación y, por otro, el diseño de la conectividad inalámbrica.

4.1 Diagrama de flujo del sistema

El código de la aplicación generado se puede observar en el Anexo I: *Código lenguaje C.*

El diagrama de flujo del sistema es el mostrado en la figura 26.

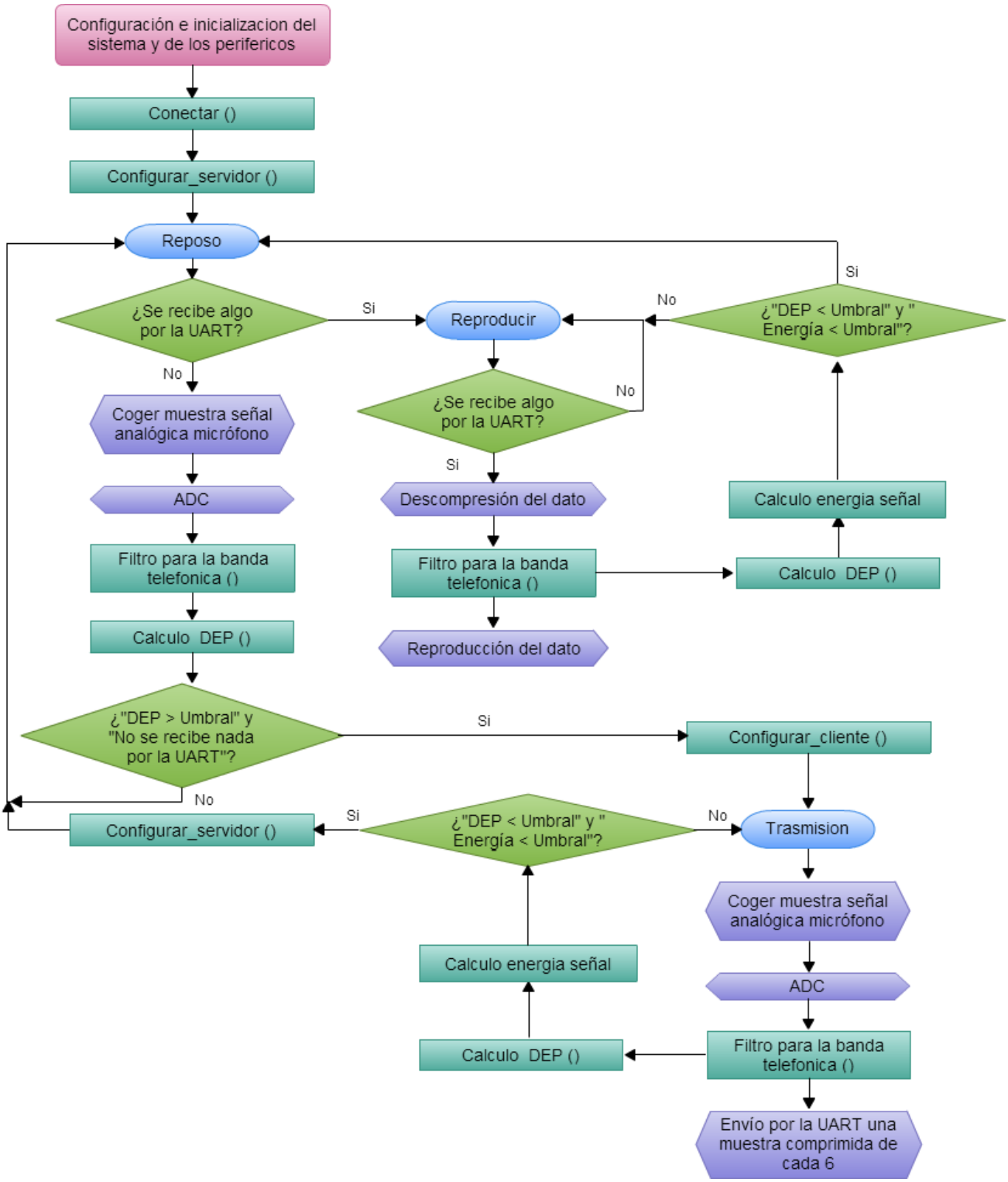

Figura 26: Diagrama de flujo del sistema

Como ha podido observarse existen 3 estados los cuales son: Reposo, Reproducir y Transmisión.

Nada más inicializarse, el sistema se configura e inicializa los periféricos para poder empezar a funcionar. Una vez pasada la configuración el sistema debe conectarse a la red inalámbrica, para ello se hace uso de la función ya explicada anteriormente en el apartado 2.5 *Módulos inalámbricos*. Una vez que el sistema se ha conectado a la red, el siguiente paso es la configuración en modo servidor la cual es la configuración por defecto de todos los dispositivos.

Cuando el sistema acaba de configurarse totalmente se pasa al estado de reposo, en el cual se comprueba si se están recibiendo datos en el dispositivo o si se está hablando sobre el dispositivo. Para ello, lo primero que se hace es la comprobación de si existe un nuevo dato proveniente de la UART en cuyo caso se pasaría al estado de Reproducir. Si no existe un dato en la UART, se pasa a comprobar si existe un hablante sobre el dispositivo. Para ello se procesan muestras de voz, con lo que se calcula la densidad espectral de potencia. Si la densidad espectral de potencia es mayor que un cierto umbral, el sistema detecta que existe un hablante por lo que se configura como cliente y pasa al estado de Transmisión.

En el estado de Transmisión el sistema coge muestras de voz y las procesa para transmitir por la UART una de cada 6 muestras (ver Anexo II: *Esquema de tiempos de reproducción*). Además con los datos que se han capturado se calcula la densidad espectral de potencia y la energía de la señal con el propósito de determinar mediante un umbral cuando existe silencio, es decir, cuando se ha dejado de hablar sobre el dispositivo.

Cabe decir, que existe un compromiso en las prestaciones del sistema el cual es un compromiso entre cuánto tiempo es silencio y cuánto tiempo es pausa del hablante.

El procesado de los datos mediante el cálculo de la DEP y de la energía de la señal se realiza tanto en el estado Transmisión como en el estado Reproducir, y sólo se vuelve al estado de Reposo en el caso que no se detecte presencia de voz.

4.2 Diseño del esquemático y componentes

Para la parte relativa a la alimentación del DSP se ha empleado la documentación de *TMS320C5535 eZdsp Module [7]* con el fin de obtener los datos necesarios para su diseño. Con respecto a la parte inalámbrica se han dejado accesibles en la placa los pines de transmisión y recepción, el pin de alimentación de 3.3V, un pin de masa, así como dos pines auxiliares por si fuera necesario el empleo de algunas señales de control.

Se ha planteado el diseño partiendo de que la alimentación va a provenir de una batería externa. Dicha batería tiene que alimentar tanto al DSP como al módulo inalámbrico.

La alimentación del DSP es de 5V tal y como indica *TMS320C5535 eZdsp Module [7]* mientras que la alimentación de los módulo suele ser de 3,3V. Por lo tanto la batería tiene que ser capaz de suministrar dicha tensión.

La batería que se ha escogido es una batería de 2200mAh y 11,1V, suficiente para alimentar durante un largo periodo de tiempo al sistema diseñado.

Posteriormente se ha elegido el conector adecuado para extraer del DSP las señales provenientes de la UART para llevarlas a la transmisión y recepción del módulo de comunicaciones. El modelo del conector es el que indica *eZdsp™ Technical Reference [2]* con referencia del fabricante FCI 87024-610LF.

Para el regulador de 5V de tensión se ha escogido el regulador 7805 un conocido regulador de tensión con las características suficientes para solucionar el problema que plantea el sistema.

Tal y como indica las hojas de características *[8]* de este regulador es necesario incorporar dos condensadores de filtrado uno a la entrada y otro a la salida que se han incorporado al diseño final. Posteriormente la salida del regulador se llevará a un conector USB para la alimentación del DSP.

Para el regulador de 3,3V se ha escogido un regulador de tensión regulable mediante un potenciómetro, con el fin de poder ajustar los 3,3V con ayuda de un osciloscopio. A la salida de éste regulador se ha incorporado un condensador de filtrado para establecer una tensión con el menor rizado posible.

Una vez definido cómo se va a alimentar al sistema, el siguiente paso es la realización del esquemático, con el propósito de tener diseñado el circuito y de tener la lista necesaria de los componentes que se incluyen en el diseño. El esquemático planteado es el que se muestra en la figura 27.

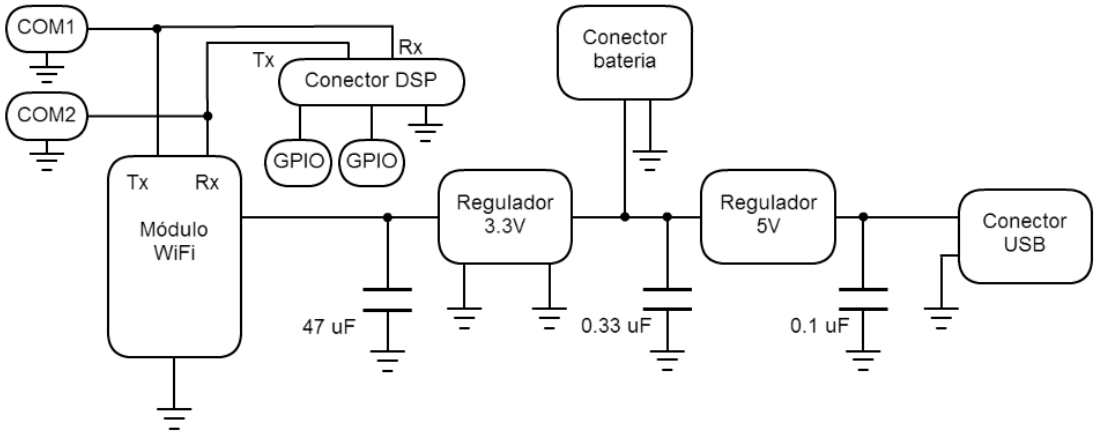

Figura 27: Esquemático de la placa de circuito impreso

Como puede observarse en la figura anterior, se han dejado accesibles mediante cuatro pines dos puertos COM para la visualización en el ordenador del intercambio de los datos establecidos, y dos GPIO para señales auxiliares para la depuración del código software o como salidas auxiliares para señales de control necesarias en ciertos módulos inalámbricos.

Los componentes necesarios para el diseño de la PCB son los que se muestran en la tabla 4.

Componente	Cantidad
Mini DC Adjustable Power Supply Buck Module Step Down Module	1
C=47uF	1
C=0.33uF	1
C=0.1uF	1
Regulador 7805	1
Conector USB	1
Conectores de dos entradas	2
Tiras de pines	1
Conector FCI 87024-610LF	1

Tabla 4: Componentes de la placa de circuito impreso

4.3 Diseño PCB

Una vez que se ha realizado el diseño del circuito, se debe crear la PCB en sí. Para ello se ha empleado el programa *EAGLE v6.5.0* con el que se ha generado el diseño mostrado en la figura 28.

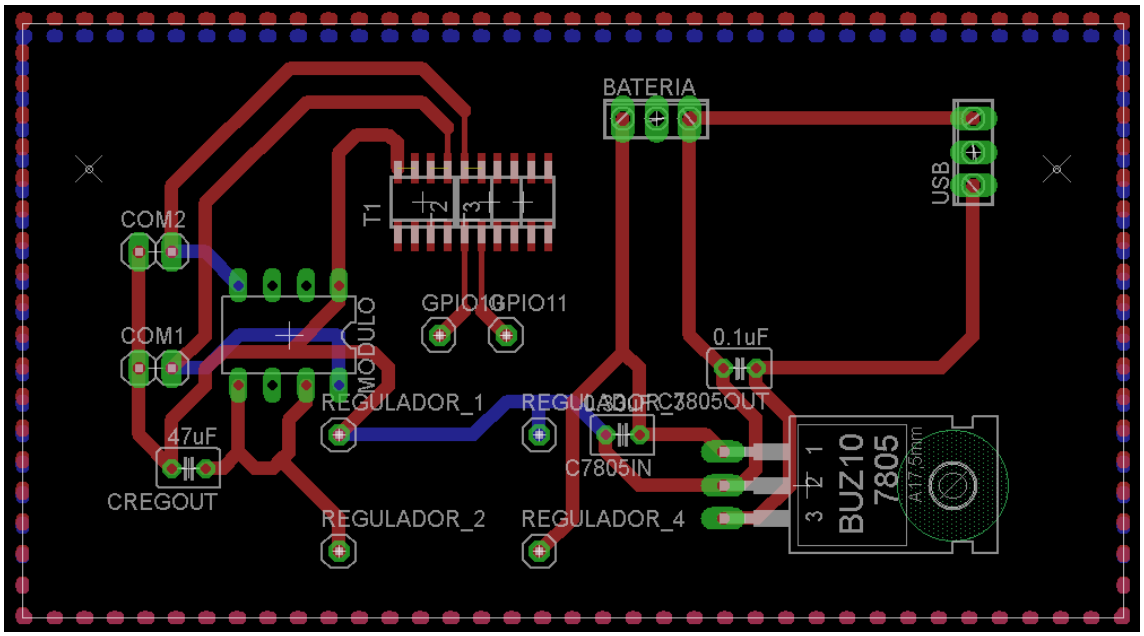

Figura 28: Diseño de la PCB en *EAGLE*

La disposición de los elementos se ha tenido en cuenta diseñando la placa de circuito impreso para que se ajuste correctamente a las dimensiones del DSP. Para asegurar más firmemente la placa al DSP se han creado dos agujeros que permiten sujetar este *shield* al DSP y no provocar que el conector se desconecte mientras el sistema esté en funcionamiento.

Además para el diseño de la PCB se ha tenido en cuenta que todos los elementos tienen que estar por arriba para poder encajar correctamente el conector y no tener problemas de espacio, por lo que se ha intentado realizar la configuración de las pistas en una sola cara de la PCB, cambiando a la segunda sólo en los casos que no quede más remedio.

4.4 Prototipado y comprobación del hardware y software

Una vez creado el hardware correspondiente se debe verificar que realmente funciona como se desea. Para ello se ha comprobado mediante el osciloscopio que las tensiones que proporcionan a la salida ambos reguladores son las correctas. Además se han hecho pruebas de funcionamiento de diferentes partes.

Para realizar el prototipado rápido del hardware sin esperar a la fabricación de la PCB [(1)], se ha utilizado una placa de prototipado de topos, en los que se ha reproducido el esquemático anterior. Los resultados pueden ser observados en la figura 29.

(1) Debido a un problema con la máquina de prototipado en PCB de la universidad, se ha decantado por utilizar una placa de topos para la fabricación de las placas.

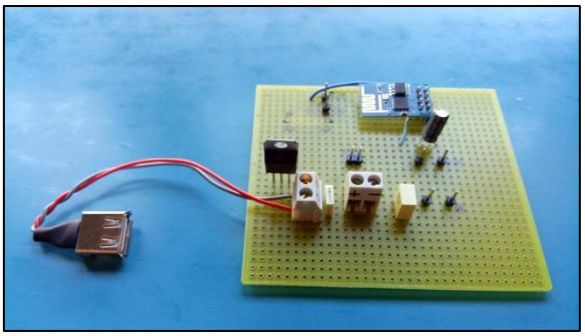

Figura 29: Sistema en placa de prototipado de topos.

Para la comprobación del software se han realizado diferentes programas de pruebas que se explican a continuación, a modo de revisión histórica del trabajo realizado hasta conseguir llegar al sistema final propuesto en este trabajo.

El sistema se puede descomponer en 5 bloques fundamentales, la configuración del DSP, la captura y reproducción del audio, el filtro, el cálculo de la DEP y la comunicación por la UART.

Para la captura de voz y la reproducción de audio se ha creado un programa el cual captura la voz a 48kHz y la reproduce en el mismo DSP pudiendo seleccionar la frecuencia de reproducción a 48kHz u 8kHz, En el *Anexo II: Esquema de tiempos de reproducción*, se presenta esta información con mayor detalle. Además en este programa se realizó el ajuste del ADC y del DAC con el fin de obtener el mejor resultado posible en la reproducción del audio.

Para la prueba del filtro, se ha creado un programa el cual filtra una función delta con el filtro de banda telefónica para la comprobación de la respuesta impulsional y se han trasladado a Matlab los resultados tal y como explica el apartado *3.1 Filtro para la banda telefónica*. Una vez comprobado el filtro, se ha aplicado al dato capturado por el micrófono en el programa anterior para poder observar los cambios que introduce en la reproducción. Otra prueba realizada en el filtro fue pasar tonos de distinta frecuencia con el fin de comprobar el correcto funcionamiento del filtrado, obteniendo los resultados esperados en función de la banda de paso y la banda de atenuación.

Para probar el cálculo de la DEP se ha creado un programa que calcula la FFT de un tono y posteriormente se aplica el cálculo de la DEP, como el módulo al cuadrado de la FFT. El cálculo de la FFT se ha realizado utilizando tanto el acelerador hardware como el cálculo por software tal y como indica el apartado *3.2 Cálculo de la Densidad Espectral de Potencia*. La FFT del tono se puede observar en la figura 30 y 31 las cuales contienen respectivamente la parte real de la FFT y la parte imaginaria. En la figura 32 se muestra la DEP como módulo al cuadrado de la FFT.

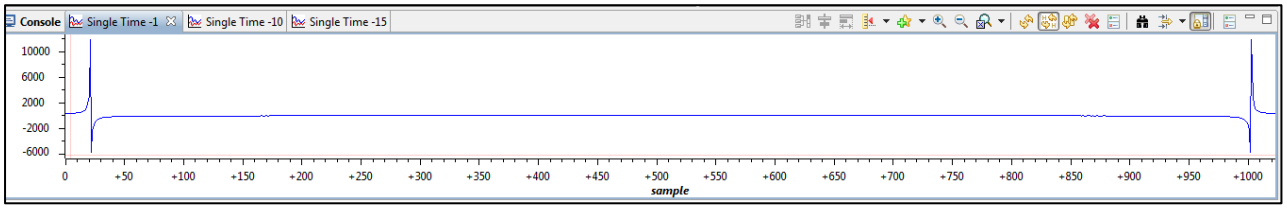

Figura 30: Parte real de la FFT del tono de prueba

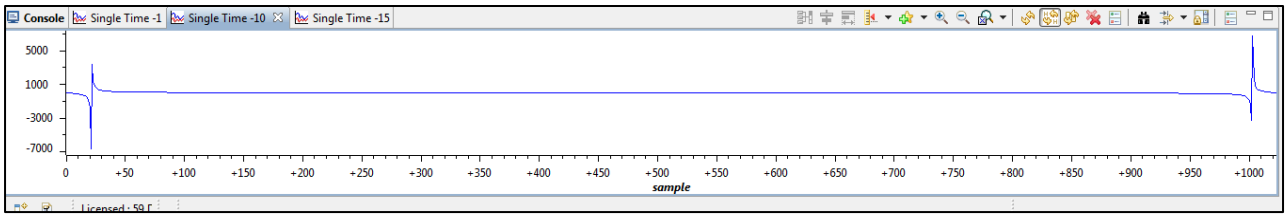

Figura 31: Parte imaginaria de la FFT del tono de prueba

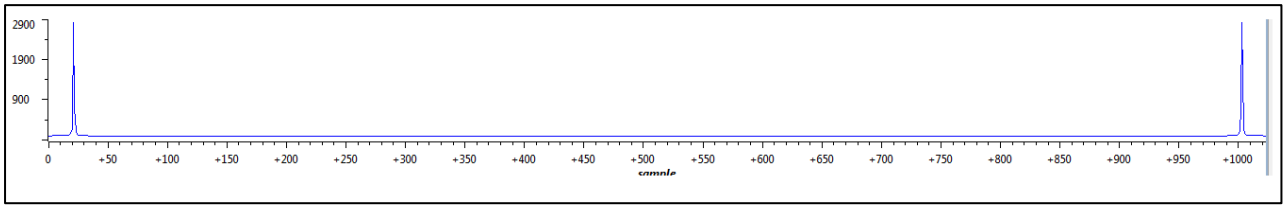

Figura 32: DEP del tono de prueba

El siguiente programa realizado para la comprobación del software fue la detección del hablante en el cual al programa anterior del filtrado de la voz y a la reproducción se le añadió el cálculo de la DEP para establecer los umbrales a partir de los cuales el sistema decide si se está hablando o no. Fue en este punto en el cual hubo que añadir la detección del silencio y se ajustó para que el sistema detectara silencio si el hablante se queda callado más de medio segundo.

Una vez verificado que todos los bloques anteriores funcionaban correctamente el siguiente paso fue la configuración de la UART para mandar el dato una vez que se detectaba que se estaba hablando. Para ello se implementaron las funciones explicadas en el apartado 3.*4 Envío por la UART*, y se realizó la comunicación entre dos DSP monitorizando en todo momento la transmisión y recepción de datos a través de los puertos COM y mediante programas como Arduino y Putty. Además se implementó un programa el cual enviaba el valor 0x30 (que se corresponde con el valor ASCII del '0') y que al recibirlo se pusiera a parpadear un led.

Verificado el funcionamiento de la configuración de la UART, se ha modificado el programa anterior de detección del hablante con el fin de incorporar esto y realizar correctamente el funcionamiento. Una vez modificado el programa se ha comprobado su funcionamiento monitorizando el intercambio de información como se ha explicado anteriormente y se ha comprobado que el sistema cumple con los objetivos del diseño.

Una vez llegados a este punto se ha intentado realizar la comunicación inalámbrica con los módulos ESP8266 disponibles y las funciones de configuración explicadas en el apartado *2.5 Módulos inalámbricos*, pero debido a los problemas de velocidad de los módulos y de las cabeceras que estos introducían no ha sido posible realizar la comunicación inalámbrica. Las soluciones que se proponen a este problema quedan recogidas en el apartado 2.5 *Módulos inalámbricos.*

5

Conclusiones

En este capítulo se hace una valoración global del sistema planteado en el trabajo. Además se hace un posible planteamiento de mejoras las cuales podrían ser implementadas en un futuro.

5.1 Características finales del sistema

Este sistema es capaz de detectar automáticamente al hablante, realizar y gestionar el control de la instalación y conectarse de manera inalámbrica, con el único inconveniente de que no se puede implementar actualmente debido a los módulos inalámbricos que se encuentran en el mercado.

El sistema ha sido planteado para una sala de conferencias, en la cual se dan por supuestas una serie de características como por ejemplo que es un lugar en la que no existe mucho ruido de fondo, se va a respetar siempre el turno de palabra entre los interlocutores, y éstos están dispuestos, en la medida de lo posible, a colaborar.

Las características del sistema que pueden presentar ciertos tipos de inconvenientes son:

- Al realizar una comunicación mediante WiFi, existe un tiempo al realizar la conexión y desconexión que puede ser elevado según el módulo inalámbrico escogido (pasar del modo servidor al modo cliente y viceversa). Por lo que desde que se detecta al hablante, hasta que el sistema manda los datos al resto puede existir un retardo bastante elevado.
- Para evitar interferir unos sistemas con otros, la sensibilidad del micrófono tiene que ser pequeña para que si dos interlocutores están cercanos no detecten los dos sistemas que se están hablando sobre ellos.
- El tiempo de detección de silencio se ha ajustado a medio segundo, el cual puede ser un tiempo pequeño si el hablante realiza pausas largas. Es decir, existe un compromiso en el tiempo de silencio en el que no se sabe si la persona ha realizado una pausa o ha dejado definitivamente de hablar.
- Al arrancar el sistema tarda un tiempo en configurarse y estar disponible.

5.2 Mejoras propuestas al sistema

Al ser una primera versión del sistema existen muchos aspectos que se podrían mejorar las cuales quedan recogidas a continuación.

- Se podría incluir un sistema de detección del hablante más robusto, es decir, introducir un algoritmo adaptativo al ruido con el fin de poder determinar el ruido de fondo que se encuentra en la sala.
- Se podría incluir la FFT por hardware si se realizara la correcta configuración de los registros tras el arranque, lo cual eliminaría carga de procesado al DSP y llevaría al sistema a ser más eficiente.
- Se podrían reducir los tiempos de conexión y desconexión en función de los módulos inalámbricos escogidos.
- Al ser una comunicación WiFi mediante UDP se podrían crear funciones alternativas y muy diversas del sistema dotándolo con conectividad a internet como por ejemplo utilizar reconocimiento de voz para pedir el turno de palabra.

- Se podría reducir el tamaño del sistema y la eficiencia escogiendo de la placa C5535 eZdsp sólo los componentes que intervienen en el sistema y realizando una PCB con estos elementos.

REFERENCIAS

[1] Texas Instruments, TMS320C5535/34/33/32 Ultra-Low Power DSP Technical Reference Manual, www.ti.com/lit/pdf/spruh87, August 2011–Revised May 2014.

[2] Spectrum Digital, TMS320C5535 eZdsp™ Technical Reference, http://support.spectrumdigital.com/boards/ezdsp5535/revc/files/ezdsp5535_TechRef_RevC.pdf , 514585-0001 Rev. A. August 2011.

[3] Texas Instruments, TLV320AIC3204 Ultra Low Power Stereo Audio Codec, http://www.ti.com/lit/ds/symlink/tlv320aic3204.pdf , SLOS602C –SEPTEMBER 2008–REVISED NOVEMBER 2014.

[4] Texas Instruments, TLV320AIC3204 Application Reference Guide, http://www.ti.com/lit/ml/slaa557/slaa557.pdf , SLAA557–November 2012.

[5] Texas Instruments, TMS320C5535/34/33/32 Fixed-Point Digital Signal Processor Silicon Revision 2.2 Silicon Errata, http://www.ti.com/lit/er/sprz373b/sprz373b.pdf , SPRZ373B–February 2012–Revised May 2014.

[6] Texas Instruments, TMS320C6000 µ-Law and A-Law Companding with Software or the McBSP, http://www.ti.com/lit/an/spra634/spra634.pdf , SPRA634 - April 2000.

[7] Spectrum Digital, TMS320C5535 EZDSP MODULE, http://support.spectrumdigital.com/boards/ezdsp5535/revc/files/ezdsp5535_Schematics_RevC.pdf, Tuesday, July 26, 2011.

[8] Fairchild Semiconductor, KA78XX/KA78XXA 3-Terminal 1A Positive Voltage Regulator, http://datasheet.eeworld.com.cn/pdf/FAIRCHILD/21191_KA7810A.pdf , Rev. 1.0.0 – 2001.

ANEXO I: Código lenguaje C

En este anexo se muestra el código en lenguaje C del sistema completo. Este capítulo contiene en los diferentes apartados las funciones: main.c, aic32044_1.c, compresión.c, descompresión.c, filtro.c, uart.c, usbstk5515_i2c.c, vectores.asm.

A.I.1 Main.c

En éste apartado se muestra el código que contiene el programa principal. Además contiene las interrupciones para transmitir y recibir datos, la función esperar(), enviar(), la función calculo_PSD(), y las funciones que configuran el módulo inalámbrico conectar(), configurar_cliente() y configurar_servidor(). Estas últimas tres funciones deberían ser modificadas con la configuración de los módulos inalámbricos correspondientes. Como ejemplo se adjunta la configuración de los módulos ESP8266, los cuales si pudieran transmitir datos en bruto (sin cabeceras) tal y como se ha descrito en el apartado *2.5 Módulos inalámbricos*, servirían para esta aplicación.

```
#include "SEmp_5515.h"
#include "usbstk5515_i2c.h"
#include "aic3204_1.h"
#include <stdio.h>
#include "stdint.h"
#include "Dsplib.h"
#include "TMS320.h"
#include "Filtro.h"
#include "compresion.h"
#include "descompresion.h"
#include "Uart.h"
#include <string.h>

#pragma DATA_SECTION(data_buf, "data_buf");
#pragma DATA_ALIGN (data_buf, 4096);
Int32 data_buf[1024];
Int32 *data = data_buf;
DATA *data_input;
DATA Re[1024], Img[1024];
LDATA PSD[1024];

int calculo_PSD(int *datos);
void enviar(char texto[]);
void esperar();
void conectar();
void configurar_servidor();
void configurar_cliente();

int16_t interrupcion_generada = 0;
int16_t interrupcion_recepcion = 0;
int16_t datoRX_filtrado;
int16_t dato_descomprimido;

int main(void) {
	int32_t suma_DEP = 0;
	uint16_t contadorBloque = 0;
	int16_t voz[1024];
	uint16_t cnt_vector = 0;
```

```
	uint16_t cnt_energia = 0;
	int32_t suma_Energia_antigua = 0;
	int32_t suma_energia = 0;
	int16_t dummy1;
	int16_t cnt_48_a_8 = 0;
	int16_t contador_salir = 0;
	typedef enum {
		reposo, transmision, reproducir
	} estados;
	estados estado = reposo;

	// Configuracion DSP y perifericos
	IO_PSRCR = 0x020;
	IO_PRCR = 0x0081;
	asm(" RPT #65535 ");
	asm(" NOP ");
	IO_PCGR1 = 0x3FBB;
	IO_PCGR2 = 0x00FF;
	PLL_CNTL1 = 0x8BE8;
	PLL_CNTL2 = 0x8000;
	PLL_CNTL3 = 0x0806;
	PLL_CNTL4 = 0x0000;
	IO_EBSR = 0x1A00

	UART_init();
	CPU_IVPD = 0x0480;
	CPU_IVPH = 0x0480;
	IODIR1=0xC000;
	IODIR2=0x03;
	IODATAOUT1=0xc000;
	IODATAOUT2=0x02;
	USBSTK5515_I2C_init();
	AIC3204_1_init();
	conectar();
	configurar_servidor();
	IODATAOUT1 =0xc000;
	IODATAOUT2=0x003;
	CPU_IER0 = 0;
	CPU_IER1 = 0;
	_enable_interrupts();
	UART_IER=0x0001;
	while ((UART_LSR&0x0081)==0x0001) {
	dummy1=UART_RBR;
	}

	//Programa
while (1) {
	switch (estado) {

	case reposo:
		IODATAOUT1 =0x0000;
		IODATAOUT2=0x0000;
		if ((UART_LSR&0x0081)==0x0001) {
			CPU_IER0 = 0x4000;
			estado = reproducir;
			cnt_vector = 0;
			contadorBloque = 0;
			suma_DEP = 1;
		dato_descomprimido=descompresion(UART_RBR);
	} else {
		CPU_IER0 = 0x8000;
		if (interrupcion_generada == 1) {
			interrupcion_generada = 0;
			voz[cnt_vector] = datoRX_filtrado;
			cnt_vector++;
			if (cnt_vector == 1024) {
```

```
                        cnt_vector = 0;
                }

                if (contadorBloque == 6000) {
                        contadorBloque = 0;
                        suma_DEP = 0;
                        suma_DEP = calculo_PSD(voz);
                }
                contadorBloque++;
                if ((suma_DEP > 0)&&(!((UART_LSR&0x0081)==0x0001))) {
                        IODATAOUT1=0xc000;
                        IODATAOUT2=0x0003;
                        configurar_cliente();
                        estado = transmision;
                        cnt_vector = 0;
                        contadorBloque = 0;
                        suma_DEP = 1;
                }
        }
    }
    break;

case transmision:
    IODATAOUT1=0x0000;
    IODATAOUT2=0x003;
    if (interrupcion_generada == 1) {
        interrupcion_generada = 0;
        if (cnt_48_a_8 == 5) {
            UART_THR=compresion(datoRX_filtrado);
            while (!((UART_LSR&0x0040)==0x0040));
            cnt_48_a_8 = 0;
        }
        cnt_48_a_8++;
        voz[cnt_vector]       = datoRX_filtrado;
        cnt_vector++;
        if (cnt_vector == 1024) {
            cnt_vector = 0;
        }

        if (contadorBloque == 24000) {
            contadorBloque = 0;
            suma_DEP = 0;
            suma_DEP = calculo_PSD(voz);
        }
        contadorBloque++;

        suma_Energia_antigua = suma_Energia_antigua
            + (((long) datoRX_filtrado * datoRX_filtrado) >> 16);
        if (cnt_energia == 24000) {
            cnt_energia = 0;
            suma_energia = suma_Energia_antigua;
            suma_Energia_antigua = 0;
        }
        cnt_energia++;
        if ((suma_DEP == 0) && (suma_energia == 0)) {
            IODATAOUT1=0xc000;
            IODATAOUT2=0x003;
            configurar_servidor();
            CPU_IER0 = 0x8000;
            estado = reposo;
            suma_Energia_antigua = 0;
            suma_energia = 0;
            cnt_vector = 0;
            contadorBloque = 0;
            suma_DEP = 0;
            while ((UART_LSR&0x0081)==0x0001) {
```

```
                        dummy1=UART_RBR;
                        }
                }
}
break;

case reproducir:
IODATAOUT1 =0xc000;
IODATAOUT2=0x0000;
CPU_IER0=0x4000;
if (interrupcion_recepcion == 1) {
        interrupcion_recepcion = 0;
        voz[cnt_vector] = dato_descomprimido;
        cnt_vector++;
        if (cnt_vector == 1024) {
                cnt_vector = 0;
        }
        if (contadorBloque == 24000) {
                contadorBloque = 0;
                suma_DEP = 0;
                suma_DEP = calculo_PSD(voz);
        }
        contadorBloque++;
        suma_Energia_antigua = suma_Energia_antigua
        + (((long) dato_descomprimido * dato_descomprimido) >> 16);
        if (cnt_energia == 24000) {
                cnt_energia = 0;
                suma_energia = suma_Energia_antigua;
                suma_Energia_antigua = 0;
        }
        cnt_energia++;
        if ((suma_DEP == 0) && (suma_energia == 0)) {
                IODATAOUT1 =0xc000;
                IODATAOUT2=0x0003;
                for(contador_salir=0;contador_salir<11;contador_salir++) {
                        esperar();}
                CPU_IER0 = 0x8000;
                estado = reposo;
                suma_Energia_antigua = 0;
                suma_energia = 0;
                cnt_vector = 0;
                contadorBloque = 0;
                suma_DEP = 0;
                while ((UART_LSR&0x0081)==0x0001) {
                        dummy1=UART_RBR;
                }
        }
}
break;
}
}
}

interrupt void ISR_I2S_rx(void) {
        int dummy;
        dummy = I2S2_W0_LSW_R;
        dummy = I2S2_W1_MSW_R;
        dummy = I2S2_W1_LSW_R;
        datoRX_filtrado= Filtro(I2S2_W0_MSW_R);
        interrupcion_generada = 1;
}

interrupt void ISR_I2S_tx(void) {
        static int contador = 0;
        I2S2_W0_LSW_W = 0x0000;
        I2S2_W1_LSW_W = 0x0000;
```

```
        if (contador == 5) {
                contador = 0;
                if ((UART_LSR&0x0081)==0x0001) {
                dato_descomprimido = descompresion(UART_RBR);
                }
        }
        contador++;
        I2S2_W0_MSW_W=Filtro(dato_descomprimido);
        I2S2_W1_MSW_W=Filtro(dato_descomprimido);
        interrupcion_recepcion = 1;
}

int calculo_PSD(int *datos) {
int i = 0;
int p = 0;
int suma_PSD = 0;
for (i = 0; i < 1024; i++) {
        data_buf[i] = ((long) datos[i] << 20) & 0xFFFF0000;
}
data_input = (DATA *) data_buf;
cfft(data_input, 1024, SCALE);
cbrev(data_input, data_input, 1024);
for (i = 0; i < 1024; i++) {
        Re[i] = data_input[2 * i];
        Img[i] = data_input[2 * i + 1];
        PSD[i] = ((long) Re[i] * Re[i] + (long) Img[i] * Img[i]) >> 16;
}
for (p = 0; p < 1024; p++) {
        suma_PSD = suma_PSD + PSD[p];
}
return suma_PSD;
}

void enviar(char texto[]) {
int valor_ASCII = 0;
int i = 0;
int longitud = strlen(texto);
for (i = 0; i < longitud; i++) {
        valor_ASCII = texto[i];
        UART_THR=valor_ASCII;
        while (!((UART_LSR&0x0040)==0x0040));
}
}

void esperar() {
int a = 0;
while (a < 1200) {
        asm(" RPT #65535 ");
        asm(" NOP ");
        a++;
}
}

void conectar() {
esperar();
enviar("AT+CWMODE=3\015\012");
esperar();
enviar("AT+CWJAP=\042MICRO\042,\042MICRO\042\015\012");
esperar();
esperar();
esperar();
esperar();
}
void configurar_servidor() {
esperar();
enviar("AT+RST\015\012");
```

```
esperar();
esperar();
esperar();
enviar("AT+CIPMUX=1\015\012");
esperar();
enviar("AT+CIPSERVER=1,9995\015\012");
esperar();
}
void configurar_cliente() {
esperar();
enviar("AT+RST\015\012");
esperar();
esperar();
esperar();
enviar("AT+CIPMODE=1\015\012");
esperar();
enviar("AT+CIPMUX=1\015\012");
esperar();
enviar("AT+CIPSTART=0,\042UDP\042,\042192.168.1.255\042,9995\015\012");
esperar();
enviar("AT+CIPSEND\015\012");
}
```

A.I.2 Aic3204_1.c

Este apartado contiene la configuración del codec AIC3204 empleado. Para ello se ha hecho uso de la función que proporciona *Digital Spectrum* y se ha modificado la función con el propósito de obtener una salida *mono* en vez de *stereo* y además proporcionar la ganancia adecuada a los conversores analógico digital y digital analógico.

```
/*
 *  Copyright 2010 by Spectrum Digital Incorporated.
 *  All rights reserved. Property of Spectrum Digital Incorporated.
 */

/*
 *  AIC3204 Test
 *
 */

#define AIC3204_I2C_ADDR 0x18

#include "SEmp_5515.h"
#include "usbstk5515_i2c.h"

/* ------------------------------------------------------------------------ *
 *                                                                          *
 *  _AIC3204_rget( regnum, regval )                                         *
 *                                                                          *
 *      Return value of codec register regnum                               *
 *                                                                          *
 * ------------------------------------------------------------------------ */
int16_t AIC3204_1_rget(uint16_t regnum, uint16_t* regval) {
       int16_t i, retcode = 0;
       uint16_t cmd[2];

       cmd[0] = regnum & 0x007F;       // 7-bit Register Address
       cmd[1] = 0;

       retcode |= USBSTK5515_I2C_write(AIC3204_I2C_ADDR, cmd, 1);
```

```
        retcode |= USBSTK5515_I2C_read(AIC3204_I2C_ADDR, cmd, 1);

        *regval = cmd[0];
        for (i = 0; i < 10; i++) {
        };            // Short delay
        return retcode;
}

/* ------------------------------------------------------------------------ *
 *                                                                          *
 *  _AIC3204_rset( regnum, regval )                                         *
 *                                                                          *
 *      Set codec register regnum to value regval                           *
 *                                                                          *
 * ------------------------------------------------------------------------ */
int16_t AIC3204_1_rset(uint16_t regnum, uint16_t regval) {
        uint16_t cmd[2];
        cmd[0] = regnum & 0x007F;       // 7-bit Register Address
        cmd[1] = regval;                // 8-bit Register Data

        return USBSTK5515_I2C_write(AIC3204_I2C_ADDR, cmd, 2);
}

/* ------------------------------------------------------------------------ *
 *                                                                          *
 *  _AIC3204_init(  )                                          *
 *                                                                          *
 *      Inicializa el codec                                 *
 *                                                                          *
 * ------------------------------------------------------------------------ */

void AIC3204_1_init(void) {
        int i;
        int ret = 0;
        IO_PCGR1=0x0000;
        IO_PCGR2=0x0000;
        /* Configure AIC3204 */
        AIC3204_1_rset(0, 0x00);  // Select page 0
        AIC3204_1_rset(1, 0x01);  // Reset codec
        for (ret = 0; ret < 1000; ret++)
                ; // Wait 1ms after reset
        AIC3204_1_rset(0, 0x01);  // Select page 1
        AIC3204_1_rset(1, 0x08);  // Disable crude AVDD generation from DVDD
        AIC3204_1_rset(2, 0x01);  // Enable Analog Blocks, use LDO power
        AIC3204_1_rset( 123,0x05 );  // Force reference to power up in 40ms
        for (ret = 0; ret < 5000; ret++)
                ;       // Wait at least 40ms
        AIC3204_1_rset(0, 0x00);  // Select page 0

        /* PLL and Clocks config and Power Up*/
        AIC3204_1_rset(27, 0x0d);  // BCLK and WCLK are set as o/p; AIC3204(Master)
        AIC3204_1_rset(28, 0x00);  // Data ofset = 0
        AIC3204_1_rset(4, 0x03); // PLL setting: PLLCLK <- MCLK, CODEC_CLKIN <-PLL CLK
        AIC3204_1_rset(6, 0x07);  // PLL setting: J=7
        AIC3204_1_rset(7, 0x06);  // PLL setting: HI_BYTE(D=1680)
        AIC3204_1_rset(8, 0x90);  // PLL setting: LO_BYTE(D=1680)
        AIC3204_1_rset(30, 0x88);
        AIC3204_1_rset(5, 0x91);  // PLL setting: Power up PLL, P=1 and R=1
        for (ret = 0; ret < 10000; ret++)
                ;  // Wait for PLL to come up
        AIC3204_1_rset(13, 0x00); // Hi_Byte(DOSR) for DOSR = 128 decimal or 0x0080
        DAC oversamppling
        AIC3204_1_rset(14, 0x80);  // Lo_Byte(DOSR) for DOSR = 128 decimal or 0x0080
        AIC3204_1_rset(20, 0x80); // AOSR for AOSR = 128 decimal or 0x0080 for
        decimation filters 1 to 6
        AIC3204_1_rset(11, 0x82);  // Power up NDAC and set NDAC value to 2
```

```
	AIC3204_1_rset(12, 0x87);  // Power up MDAC and set MDAC value to 7
	AIC3204_1_rset(18, 0x87);  // Power up NADC and set NADC value to 7
	AIC3204_1_rset(19, 0x82);  // Power up MADC and set MADC value to 2

	/* DAC ROUTING and Power Up */
	AIC3204_1_rset(0, 0x01);  // Select page 1
	AIC3204_1_rset(12, 0x08);  // LDAC AFIR routed to HPL
	AIC3204_1_rset(13, 0x08);  // RDAC AFIR routed to HPR
	AIC3204_1_rset(0, 0x00);  // Select page 0
	AIC3204_1_rset(64, 0x02);  // Left vol=right vol
	AIC3204_1_rset(65, 0x20);  // Left DAC gain to 0dB VOL; Right tracks Left
	(Maximo 0x30)
	AIC3204_1_rset(63, 0xd4);  // Power up left,right data paths and set channel
	AIC3204_1_rset(83, 0x28);    // Control de volumen del conversor ADC en el
	canal izquierdo
	AIC3204_1_rset(84, 0x28); // Control de volumen del conversor ADC en el canal
	derecho (Maximo 0x28)
	AIC3204_1_rset(0, 1);      // Select page 1
	AIC3204_1_rset(0, 0x01);  // Select page 1
	AIC3204_1_rset(16, 0x00);  // Unmute HPL , 0dB gain
	AIC3204_1_rset(17, 0x00);  // Unmute HPR , 0dB gain
	AIC3204_1_rset(9, 0x30);  // Power up HPL,HPR
	for (ret = 0; ret < 100; ret++)
		;  // Short delay

	/* ADC ROUTING and Power Up*/
	AIC3204_1_rset(0, 0x01);  // Select page 1
	AIC3204_1_rset(51, 0x48); // Power up MICBIAS with AVDD (0x40) or LDOIN (0x48)
	AIC3204_1_rset(52, 0x30);  // STEREO 1 Jack
	AIC3204_1_rset(55, 0x30);  // IN2_R to RADC_P through 40 kohmm
	AIC3204_1_rset(54, 0x03);
	AIC3204_1_rset(57, 0xc0);
	AIC3204_1_rset(59, 0x00);  // MIC_PGA_L unmute
	AIC3204_1_rset(60, 0x00);  // MIC_PGA_R unmute
	AIC3204_1_rset(0, 0x00);  // Select page 0
	AIC3204_1_rset(81, 0xc0);  // Powerup Left and Right ADC
	AIC3204_1_rset(82, 0x00);  // Unmute Left and Right ADC
	AIC3204_1_rset(0, 0x00);  // Select page 0
	for (i = 0; i < 200; i++);            // Short delay

	/* I2S settings */
	I2S2_SRGR = 0x0;
	I2S2_CR = 0x8010;
	I2S2_ICMR = 0x24;
}
```

A.I.3 Compresion.c

En este apartado se ha llevado a cabo la implementación de la función de compresión utilizando la ley-A, la cual permite obtener un dato comprimido de 8 bits a partir de uno de 16 bits.

```
unsigned int compresion(int input) {
	//Ley-A
	int segmento=0;
	unsigned int i=0, signo=0, magnitud=0;
	unsigned int output=0, absoluto=0, temp=0;
	signo = (input >= 0) ? 0 : 1;
	temp = absoluto = (abs(input) >> 3);
	for (i = 0; i < 16; i++) {
		output = temp & 0x8000;
		if (output)
```

```
                        break;
                temp <<= 1;
        }
        segmento=(11-i);
        if (segmento <= 0) {
                segmento = 0;
                magnitud = (absoluto >> 1) & 0x0F;
        } else
                magnitud = (absoluto >> segmento) & 0x0F;
        segmento <<= 4;
        output = segmento + magnitud;
        if (absoluto > 4095)
                output = 0x7F;
        if (signo)
                return output ^= 0x80;
        else
                return output;
}
```

A.I.4 Descompresion.c

En este apartado se ha llevado a cabo la implementación de la función de descompresión utilizando la ley-A, la cual permite obtener un dato descomprimido de 16 bits a partir de uno de 8 bits comprimido previamente.

```
int descompresion(unsigned int entrada) {
        //Ley-A
        unsigned int signo=0, segmento=0, magnitud=0;
        signo = (entrada & 0x80) >> 7;
        segmento = (entrada & 0x70) >> 4;
        magnitud = entrada & 0x0F;
        magnitud <<= 1;
        if (!segmento)
                magnitud += 1;
        else {
                magnitud += 33;
                magnitud <<= segmento-1;
        }
        magnitud=(magnitud<<3);
        if (signo)
                return -magnitud;
        else
                return magnitud;
}
```

A.I.5 Filtro.c

En este apartado se ha implementado la función que realiza el filtro descrito en el apartado *3.1 Filtro para la banda telefónica*, el cual es un filtro paso banda entre 300hz y 3 kHz.

```
int Filtro(int entrada) {
        int salida_filtro;
        const int g1 = 26586; // <16.16>
        const int g2 = 26586; // <16.16>
        const int g3 = 22634; // <16.16>
        const int g4 = 22634; // <16.16>
        const int g5 = 20464; // <16.17>
```

```
//Primera Etapa
const int b1_2 = -28416; // b1_2 <16.14>
const int a1_2 = -29772; // a1_2 <16.14>
const int a1_3 = 31662; // a1_3 <16.15>

long int x0_1, x1_1, x2_1;
long int salida_intermedia_1;
int salida_1;
long int y1_1, y2_1;
static long int acum1_1 = 0, acum2_1 = 0;

x0_1 = (((long) entrada) << 15); //<16.15> (<<15) = <32.30>
       x1_1 = ((((long) entrada) * b1_2) << 1); // <16.15>*<16.14> = <32.29>
(<<1) = <32.30>
x2_1 = x0_1; // <32.30>
salida_intermedia_1 = (x0_1 + acum1_1) + (0x00008000); // <32.30> redondeado a
16 bits
salida_1 = (salida_intermedia_1 >> 16); // <16.14> del truncado anterior con
16 bits
y1_1 = (((long) salida_1 * (-a1_2)) << 2); // <16.14>*<16.14>= <32.28> (<<2)=
<32.30>
y2_1 = (((long) salida_1 * (-a1_3)) << 1); // <16.14>*<16.15>= <32.29> (<<1)=
<32.30>
acum1_1 = x1_1 + y1_1 + acum2_1; // <32.30> + <32.30> + <32.30> = <32.30>
acum2_1 = x2_1 + y2_1; // <32.30> + <32.30> = <32.30>

//Segunda Etapa
const int b2g1 = -26554; // <16.15>
const int a2_2 = -32601; // a2_2 <16.14>
const int a2_3 = 32576; // a2_3 <16.15>

long int x0_2, x1_2, x2_2;
long int salida_intermedia_2;
int salida_2;
long int y1_2, y2_2;
static long int acum1_2 = 0, acum2_2 = 0;

x0_2 = ((((long) salida_1) * g1) << 1); // <16.14>*<16.16> = <32.30> (<<1)=
<32.31>
x1_2 = ((((long) salida_1) * b2g1) << 2); // <16.14>* <16.15> = <32.29> (<<2)
= <32.31>
x2_2 = x0_2; //<32.31>
salida_intermedia_2 = (x0_2 + acum1_2) + (0x00008000); // <32.31> redondeando
a 16 bits
salida_2 = (salida_intermedia_2 >> 16); // <16.15> del truncado anterior con
16 bits
y1_2 = (((long) salida_2 * (-a2_2)) << 2); // <16.15>*<16.14> = <32.29> (<<2)
= <32.31>
y2_2 = (((long) salida_2 * (-a2_3)) << 1); // <16.15>*<16.15> = <32.30> (<<1)
= <32.31>
acum1_2 = x1_2 + y1_2 + acum2_2; // <32.31> + <32.31> + <32.31> = <32.31>
acum2_2 = x2_2 + y2_2; //<32.31> + <32.31> = <32.31>

//Tercera Etapa
const int b2g2 = -20030; // <16.15>
const int a3_2 = -29437; // a3_2 <16.14>
const int a3_3 = 29146; // a3_3 <16.15>

long int x0_3, x1_3, x2_3;
long int salida_intermedia_3;
int salida_3;
long int y1_3, y2_3;
static long int acum1_3 = 0, acum2_3 = 0;
```

```
x0_3 = (((((long) salida_2) * g2) + (0x00000001)) >> 1); // <16.15>*<16.16> =
<32.31> (>>1)= <31.30>
x1_3 = (((long) salida_2) * b2g2); // <16.15>* <16.15> = <32.30>
x2_3 = x0_3; //<31.30>
salida_intermedia_3 = (x0_3 + acum1_3) + (0x00008000); // <32.30>
salida_3 = (salida_intermedia_3 >> 16); // <16.14>
y1_3 = (((long) salida_3 * (-a3_2)) << 2); // <16.14>*<16.14> = <32.28> (<<2)
= <32.30>
y2_3 = (((long) salida_3 * (-a3_3)) << 1); // <16.14>*<16.15> = <32.29> (<<1)
= <32.30>
acum1_3 = x1_3 + y1_3 + acum2_3; // <32.30> + <32.30> + <32.30> = <32.30>
acum2_3 = x2_3 + y2_3; //<31.30> + <32.30> = <32.30>

//Cuarta Etapa
const int b2g3 = -22620; // <16.15>
const int a4_2 = -32160; // a4_2 <16.14>
const int a4_3 = 31768; // a4_3 <16.15>

long int x0_4, x1_4, x2_4;
long int salida_intermedia_4;
int salida_4;
long int y1_4, y2_4;
static long int acum1_4 = 0, acum2_4 = 0;

x0_4 = (((long) salida_3) * g3); // <16.14>*<16.16> = <32.30>
x1_4 = ((((long) salida_3) * b2g3) << 1); // <16.14>* <16.15> = <32.29> (<<1)
= <32.30>
x2_4 = x0_4; //<32.30>
salida_intermedia_4 = (x0_4 + acum1_4) + (0x00008000); // <32.30> redondeando
a 16 bits
salida_4 = (salida_intermedia_4 >> 16); // <16.14> del truncado anterior con
16 bits
y1_4 = (((long) salida_4 * (-a4_2)) << 2); // <16.14>*<16.14> = <32.28> (<<2)
= <32.30>
y2_4 = (((long) salida_4 * (-a4_3)) << 1); // <16.14>*<16.15> = <32.29> (<<1)
= <32.30>
acum1_4 = x1_4 + y1_4 + acum2_4; // <32.30> + <32.30> + <32.30> = <32.30>
acum2_4 = x2_4 + y2_4; //<32.30> + <32.30> = <32.30>

//Quinta Etapa
const int b2g4 = -22634; // <16.16>
const int a5_2 = -30587; // a5_2 <16.14>
const int a5_3 = 29208; // a5_3 <16.15>

long int x0_5, x2_5;
long int salida_intermedia_5;
int salida_5;
long int y1_5, y2_5;
static long int acum1_5 = 0, acum2_5 = 0;

x0_5 = (((((long) salida_4) * g4)+(0x0000001))>>1); // <16.14>*<16.16> =
<32.30> (>>1) = <31.29>
x2_5 = (((((long)salida_4)*b2g4)+(0x00000001))>>1); // <16.14> *
<16.16>(22634) = <32.30> (>>1) = <31.29>
salida_intermedia_5 = (x0_5 + acum1_5) + (0x00008000); // <32.29> redondeando
a 16 bits
salida_5 = (salida_intermedia_5 >> 16); // <16.13> del truncado anterior con
16 bits
y1_5 = (((long) salida_5 * (-a5_2)) << 2); // <16.13>*<16.14> = <32.27> (<<2)
= <32.29>
y2_5 = (((long) salida_5 * (-a5_3)) << 1); // <16.13>*<16.15> = <32.28> (<<1)
= <32.29>
acum1_5 =y1_5 + acum2_5; // <32.29> + <32.29> = <32.29>
acum2_5 = x2_5 + y2_5; //<31.29> + <32.29> = <32.29>

//Salida final
```

```
        long int salida_intermedia_total;

        salida_intermedia_total=((((long)salida_5)*g5)>>16);      //<16.13> * <16.17> =
        <32.30> (>>16) = <16.14>
        salida_filtro = salida_intermedia_total;
        return salida_filtro;
}
```

A.I.6 Uart.c

En este apartado se han configurado los parámetros necesarios para la utilización de la UART a velocidad de 115200 baudios. Además se han seleccionado las siguientes características: un bit de *start*, 8 bits de datos, un bit de *stop* y no paridad.

```
/*
 * Uart.c
 *
 *  Created on: 12/05/2015
 *      Author: Ivan
 */

#include "SEmp_5515.h"
#include "Uart.h"
void UART_init(void) {

UART_PWREMU_MGMT = 0x0000; //Tx y Rx deshabilitados y en estado de reset
UART_DLL=0x0036;//Divisor =54 para tener baud rate 115200
divisor=f_clk_dsp/(baud_rate_desired*16)
UART_DLH=0x0000;//0x028B para 9600 y 36 para 115200
UART_FCR = 0x0001;//FIFO mode
UART_FCR=0x0007;
UART_LCR=0x0003;//Palabras de 8 bits, 1 bit stop, no paridad, no stick paridad, no
break control
UART_MCR=0x0000;//0x0010 Loopback activado (0x0000 deshabilita loopback y habilita
las señales de los pines)
UART_PWREMU_MGMT=0x6000;//Tx y Rx habilitados
}
```

A.I.7 Usbstk5515_i2c.c

En este apartado se ha hecho uso de la función proporcionada por *Digital Spectrum* la cual configura el bus de control I^2C. El cual es necesario para el intercambio de información de control y configuración entre el DSP y el códec AIC3204.

```
/*
 *  Copyright 2010 by Spectrum Digital Incorporated.
 *  All rights reserved. Property of Spectrum Digital Incorporated.
 */

/*
 *  I2C implementation
 *
 */
#include "usbstk5515_i2c.h"

int32_t i2c_timeout = 0x0fff;
```

```
/* ------------------------------------------------------------------------ *
 *                                                                          *
 *  _I2C_init( )                                                            *
 *                                                                          *
 *      Enable and initalize the I2C module                                 *
 *      The I2C clk is set to run at 20 KHz                                 *
 *                                                                          *
 * ------------------------------------------------------------------------ */
int16_t USBSTK5515_I2C_init( )
{
    I2C_MDR = 0x0400;// Reset I2C
    I2C_PSC   = 15; // Config prescaler for 100MHz
    I2C_CLKL  = 25; // Config clk LOW for 100kHz
    I2C_CLKH  = 25; // Config clk HIGH for 100kHz
    I2C_MDR   = 0x0420; // Release from reset; Master, Transmitter, 7-bit address
    return 0;
}

/* ------------------------------------------------------------------------ *
 *                                                                          *
 *  _I2C_close( )                                                           *
 *                                                                          *
 * ------------------------------------------------------------------------ */
int16_t USBSTK5515_I2C_close( )
{
    I2C_MDR = 0; // Reset I2C
    return 0;
}

/* ------------------------------------------------------------------------ *
 *                                                                          *
 *  _I2C_reset( )                                                           *
 *                                                                          *
 * ------------------------------------------------------------------------ */
int16_t USBSTK5515_I2C_reset( )
{
    USBSTK5515_I2C_close( );
    USBSTK5515_I2C_init( );
    return 0;
}

/* ------------------------------------------------------------------------ *
 *                                                                          *
 *  _I2C_write( i2c_addr, data, len )                                       *
 *                                                                          *
 *      I2C write in Master mode                                            *
 *                                                                          *
 *      i2c_addr    <- I2C slave address                                    *
 *      data        <- I2C data ptr                                         *
 *      len         <- # of bytes to write                                  *
 *                                                                          *
 * ------------------------------------------------------------------------ */
int16_t USBSTK5515_I2C_write( uint16_t i2c_addr, uint16_t* data, uint16_t len )
{
    int16_t timeout, i;
        I2C_CNT = len;                  // Set length
        I2C_SAR = i2c_addr;             // Set I2C slave address
        I2C_MDR = MDR_STT               // Set for Master Write
                  | MDR_TRX
                  | MDR_MST
                  | MDR_IRS
                  | MDR_FREE;
        for ( i = 0 ; i < 100 ; i++ ){ };           // Short delay

        for ( i = 0 ; i < len ; i++ )
```

```
        {
           I2C_DXR = data[i];            // Write
            timeout = 0x510;             // I2C_timeout = 1ms;
            do
            {
                if ( timeout-- < 0  )
                {
                    USBSTK5515_I2C_reset( );
                    return -1;
                }
            } while ( ( I2C_STR & STR_XRDY ) == 0 );// Wait for Tx Ready
        }

        I2C_MDR |= MDR_STP;             // Generate STOP

        for ( i = 0 ; i < 1000 ; i++ ){ };            // Short delay;
        return 0;
}

/* ------------------------------------------------------------------------ *
 *                                                                          *
 *  _I2C_read( i2c_addr, data, len )                                        *
 *                                                                          *
 *      I2C read in Master mode                                             *
 *                                                                          *
 *      i2c_addr    <- I2C slave address                                    *
 *      data        <- I2C data ptr                                         *
 *      len         <- # of bytes to write                                  *
 *                                                                          *
 *      Returns:    0: PASS                                                 *
 *                 -1: FAIL Timeout                                         *
 *                                                                          *
 * ------------------------------------------------------------------------ */
int16_t USBSTK5515_I2C_read( uint16_t i2c_addr, uint16_t* data, uint16_t len )
{
    int32_t timeout, i;
    I2C_CNT = len;                  // Set length
    I2C_SAR = i2c_addr;             // Set I2C slave address
    I2C_MDR = MDR_STT               // Set for Master Read
              | MDR_MST
              | MDR_IRS
              | MDR_FREE;
    for ( i = 0 ; i < 10 ; i++ ){ };            // Short delay
    for ( i = 0 ; i < len ; i++ )
    {
        timeout = i2c_timeout;
        //Wait for Rx Ready
        do
        {
            if ( timeout-- < 0 )
            {
                USBSTK5515_I2C_reset( );
                return -1;
            }
        } while ( ( I2C_STR & STR_RRDY ) == 0 );// Wait for Rx Ready

        data[i] = I2C_DRR;          // Read
    }
    I2C_MDR |= MDR_STP;             // Generate STOP
    for ( i = 0 ; i < 10 ; i++ ){ };            // Short delay
    return 0;
}
```

A.I.8 Vectores.asm

En este apartado se ha incluido la configuración del fichero vectores.asm el cual es necesario para la compilación del programa principal.

```
;vectors.asm
;
        .global _c_int00, _ISR_I2S_rx, _ISR_I2S_tx

        .global _Reset  ; Es necesario incluirla con este nombre

            .sect "vectores"
_Reset:        .ivec _c_int00,USE_RETA
NMI:           .ivec    dummy_isr             ; Nonmaskable Interrupt
INT0:          .ivec    dummy_isr             ; External User Interrupt #0
INT1:          .ivec    dummy_isr             ; External User Interrupt #1
TINT0:         .ivec    dummy_isr             ; Timer #0 / Software   Interrupt #4
PROG0:         .ivec    dummy_isr             ; Programmable 0 Interrupt
UART:          .ivec    dummy_isr             ; IIS #1 Receive Interrupt
PROG1:         .ivec    dummy_isr             ; Programmable 1 Interrupt
DMA:           .ivec    dummy_isr             ; DMA Interrupt
PROG2:         .ivec    dummy_isr             ; Programmable 2 Interrupt
COPROCFFT:     .ivec    dummy_isr             ; Coprocessor FFT Module Interrupt
PROG3:         .ivec    dummy_isr             ; Programmable 3 Interrupt
LCD:           .ivec    dummy_isr             ; LCD Interrupt
SARADC:        .ivec    dummy_isr             ; SAR ADC Interrupt
XMIT2:         .ivec    _ISR_I2S_tx           ; I2S2 Tx Interrupt
RCV2:          .ivec    _ISR_I2S_rx           ; I2S2 Rx Interrupt
XMIT3:         .ivec    dummy_isr             ; I2S3 Tx Interrupt
RCV3:          .ivec    dummy_isr             ; I2S3 Rx Interrupt
RTC:           .ivec    dummy_isr             ; RTC interrupt
SPI:           .ivec    dummy_isr             ; SPI Receive Interrupt
USB:           .ivec    dummy_isr             ; USB Transmit Interrupt
GPIO:          .ivec    dummy_isr             ; GPIO Interrupt
EMIF:          .ivec    dummy_isr             ; EMIF Interrupt
I2C:           .ivec    dummy_isr             ; IIC interrupt
BERRIV:
IV24:          .ivec    dummy_isr             ; Bus error interrupt
DLOGIV:
IV25:          .ivec    dummy_isr             ; Data log (RTDX) interrupt
RTOSIV:
IV26:          .ivec    dummy_isr             ; Real-time OS interrupt
IV27:          .ivec    dummy_isr             ; General-purpose software-only
interrupt
IV28:          .ivec    dummy_isr             ; General-purpose software-only
interrupt
IV29:          .ivec    dummy_isr             ; General-purpose software-only
interrupt
IV30:          .ivec    dummy_isr             ; General-purpose software-only
interrupt
IV31:          .ivec    dummy_isr             ; General-purpose software-only
interrupt

            .text           ; El código siguiente se incluye en la seccion .text
dummy_isr   B dummy_isr     ; Bucle infinito, bloquea el DSP en caso de error
                            ;   en las interrpciones

.end
```

ANEXO II: Esquema de tiempos de reproducción.

Para explicar con más detalle la implementación de la captura y reproducción del audio se adjunta la figura 33: Esquema de tiempos de reproducción, en la que se explica cómo se realiza el procesado y a qué velocidad en cada una de las partes del sistema.

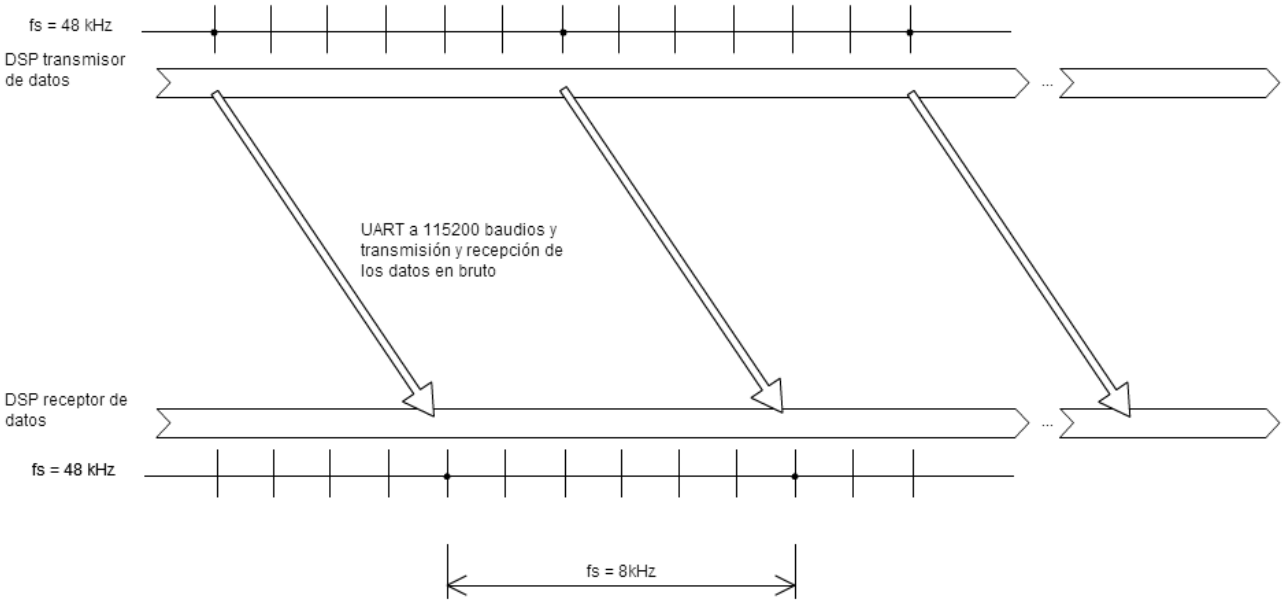

Figura 33: Esquema de tiempos de reproducción

Como se puede observar en la figura 33, el DSP transmisor de datos coge las muestras de voz a 48 kHz pero solamente envía por la UART una de cada seis muestras, por lo que la frecuencia aparente es de 8kHz. Estos datos son enviados a través de la UART y se reproducen en el DSP receptor. El DSP receptor genera las interrupciones a 48 kHz, en las cuales cada 6 muestras, es decir cada 8kHz, comprueba si existe un nuevo dato en la UART en cuyo caso actualiza la variable que contiene al dato y lo reproduce.

Como se está transmitiendo y reproduciendo a 8kHz basta con configurar la UART a 115200 baudios, ya que la velocidad mínima necesaria para reproducir a frecuencia de 8kHz sería de 80000 baudios.

El filtrado del dato como se ha comentado en el apartado 3.1 *Filtro para la banda telefónica* se realiza a 48 kHz y aunque no todos los datos en transmisión son enviados por la UART, el filtrado sí que se realiza a 48 kHz.

En la reproducción del audio se ha tenido en cuenta que una vez que el dato llega al sistema, éste está presente durante 8 interrupciones de 48kHz, lo cual genera un sobremuestreo de la señal y se debe filtrar para seguir cumpliendo el teorema de Nyquist. El filtro empleado es el del apartado 3.1 *Filtro para la banda telefónica*.